N I M : A NETWORK INTERFACE MACHINE

FOR DISTRIBUTED COMPUTER NETWORKS

by P.J. Chappell B Sc

A Thesis submitted in partial fulfilment
of the requirements for the Degree of
Master of Science in Computer Science
at the University of Canterbury
1981

ABSTRACT

The Network Interface Machine (NIM) is a microprocessor-based interface used to connect diverse computers and terminals to a communications network. The purpose of using the NIM, as contrasted to direct connection, is to offload the network interface mechanisms completely from the user devices and the underlying data communication facility. This offloading provides a portable, simplified, and easily implemented, network interface which is adaptable to computers, terminals and networks with diverse characteristics.

This thesis defines the architecture, internal design and implementation approach of the NIM, and the high-level description language used to specify the characteristics of a particular NIM implementation. NIM software is produced from the high-level description by program-generation techniques.

The thesis surveys data communication concepts, network architectures and implementation techniques. The benefits of general-purpose local computer networks are discussed, and the requirements that such networks must satisfy are identified. The NIM concept is proposed as an answer to these requirements. The architecture, protocols, implementation and description language for the areas of network control, terminal virtualisation, and external access interfaces are described in detail.

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to those people who have helped me throughout the preparation of this thesis.

I wish to thank my supervisor, Dr M A MacLean, for his guidance, and particularly for his comments and suggestions during the writing of this thesis.

I particularly wish to thank my employers, Sperry Univac, for their generous support over the entire project, and the management of the Wanganui Computer Centre for their cooperation and leave of absence enabling the completion of this project. I would also like to thank my typist, Suzanne, for her helpful and thoroughly professional assistance.

My gratitude to my friends, Ross and Mary, for their loyal friendship during my years at Canterbury, and their motivation and selfless hospitality during the preparation of this dissertation, will never be forgotten.

Finally, I would like to take this opportunity to thank my parents for their consistent generosity and encouragement throughout my education ... A very special 'thank you' Mum and Dad.

CONTENTS

CONTENTS continued ...

CONTENTS continued ...

CONTENTS continued ...

CONTENTS continued ...

CONTENTS - FIGURES

CONTENTS - FIGURES continued ...

1. INTRODUCTION

The current emphasis on distributed computer networks is but the latest phase in an evolutionary trend to bring computer access directly to the user. This trend began with the connection of a teleprinter directly to the computer and rudimentary software to allow a single user to interact directly with a running program. From this, time-sharing systems were developed which provided concurrent access to many users while efficiently utilising the computer resources.

As the use of terminal access networks increased, an awareness developed of the advantages of allowing computers to communicate not only with terminals, but also with other computers. Although at first each computer appeared to the other simply as a terminal, this started a trend towards the distribution of computational power away from a central monolithic complex.

It is not difficult to provide access to a group of compatible machines. In the simplest case, a terminal can be switched between individual physical circuits to each machine. An inter-machine link enables the convenient transfer of programs and data via utility programs, rather than the enforced use of physical media such as magnetic tape. More sophisticated software can provide generalised remote access and the possibility of workload and resource sharing. Some manufacturers already provide, and most are developing, such facilities.

It is much more difficult to establish communication between different computers, even without considering any degree of cooperation. Effective communication requires a generalised data transport network which operates independently of any particular connected computer, and provides an easily adapted interface to all types of computers and terminals. The concept is not new; it had its birth in the ARPANET which connects diverse host machines in educational and research establishments throughout the United States, and has been in operation for over a decade. However, the unavailability of suitable 'off the shelf' hardware and software to interconnect diverse systems, and the high cost of custom development, has prohibited the implementation of all but a few experimental networks.

As communications technology continues to evolve, new transmission methods and protocols will be utilised, but the functionality of the network will, to a great extent, be determined by the design of the network interface. The network interface capabilities and the methods of their implementation primarily determine the 'personality' of the network - they affect the ease of use, transparency, vendor independence, cost and flexibility of the network.

The network should aim to provide a 'friendly' interface, so that the attached devices may be unified into a complete system with minimised impact on each device. If the network interface is isolated as much as possible from the user devices and the underlying data transport services, the network will have the required versatility and capabilities to provide transparent interconnections for a wide range of devices and data transport architectures.

The microprocessor based communications processor has the potential to become an 'off the shelf' building block for distributed communications networks, in the same manner as modems and multiplexors are for terminal access networks today. However, before this potential can be realised, communications processor software systems must be developed possessing the attributes of modularity, adaptability, transparency and portability.

Modularity refers to the ability to configure a communications processor tailored to a particular physical environment by implementing functions using differing hardware/software techniques to provide the appropriate performance level at minimum cost.

Adaptability refers to the ability to change the logical functions of the communications processor to support different line protocols and terminal types.

Transparency implies that a communications network connection between two end-points should be indistinguishable from a physical link. In particular, the network itself must establish and maintain logical connections internally, and must adapt its interface to the external devices.

Portability implies that the process of transferring the communications software generation system between computers must be substantially less expensive than its initial implementation. Ideally the generation system should run on any reasonably-sized minicomputer with adequate mass storage.

This thesis investigates the design and implementation of a Network Interface Machine (NIM) which possesses the abovementioned attributes. The NIM implements a generalised interface between diverse terminals and computers, and a data transport facility. The interface functions are specified by a high-level language description of the terminal and communications link attributes, which directs a specially-developed 'program generator' or compiler to generate a customised communications processor operating system.

The following chapters examine the development and characteristics of present communications sytems, resulting in a set of design requirements for the Network Interface Machine. An overall architectural framework is developed, leading to a detailed description of the architecture, protocols, implementation and description language for the network control, terminal virtualisation, and external access interface components.

2. BASIC NETWORK CONCEPTS AND EVOLUTION

Data communications systems have gradually evolved to provide increasing remote accessibility to computer systems utilising both improved hardware technology and more efficient data transmission techniques. This chapter traces the architectural evolution of terminal access networks in terms of their physical configuration and logical structure, and their expansion to multi-domain and distributed networks. This leads to the concept of an independent data transportation network providing a generalised communication facility between a set of terminal nodes which have peer, rather than hierarchical, relationships with each other.

2.1 DEFINITIONS

The following basic concepts are defined here, and used throughout.

A *Protocol* refers to the agreements between two parties involved in the interchange of information relating to the format and meaning of control messages, and the sequence in which they are to be exchanged between the parties.

A *Node* is a physical entity within the network which may be addressed by a unique identifier. Its name is derived from graph theory where nodes are junction points formed by connecting lines.

A *Host* is a provider of computer facilities which may be shared between many users. Physically it often consists of a large mainframe computer system.

A *Station* is the logical point of attachment between a terminal or computer and the network.

A *Terminal* consists of one or more input/output devices (keyboard, display, printer etc.). Each device provides control functions which determine the formatting of the data, in addition to the basic input and output functions.

A *Device Control Protocol* refers to the set of command formats that are transmitted to a device to initiate particular control functions.

A *Data Link Control Protocol* is a protocol which manages the operation of a data link between nodes. It regulates the initiation, checking, and, if necessary, retransmission of each message to ensure efficient and reliable data transmission. Two well known examples are BISYNC (BSC) and HDLC.

An *End-To-End Control Protocol* is a protocol which controls the overall transfer of information between the source and destination nodes. It ensures overall data integrity, controls the rate of data flow, and may provide for the segmentation of user messages into network data units (messages of a fixed maximum size) commonly known as packets.

An *Architecture* refers to the overall structure of the communications system. It defines the logical structures and relationships of all functional components within the system and establishes a set of rules and guidelines which determine how the logical structures and functions are applied to the physical system through specific structures, interfaces and protocols.

2.2 EVOLUTION OF TERMINAL ACCESS NETWORKS

This section traces the evolution of terminal access networks into a layered software and hardware hierarchy, and the techniques developed to achieve the efficient utilisation of transmission facilities.

2.2.1 Logical Structures

In the earliest systems providing remote access communications, responsibility for the management of the communication link and the external device lay solely within the province of the applications program. This resulted in a multiplicity of incompatible data link control protocols, device control protocols and end-to-end protocols. Often these were considered as a single problem. Because of the diversity it was impossible to handle all link control functions within a single communications program; rather they were embedded within each user program, together with device control functions which formatted data and interpreted control characters in a unique manner for each device. Since a user program was intimately concerned with the characteristics of particular devices, any change in the device or data link protocol required corresponding changes in each program. This also had the effect of restricting the use of lines and terminals to only those programs which implemented the data link and device control protocols for those particular devices. The use of a single line for different types of terminals or even different applications was generally impossible because the devices and user programs themselves determined the characteristics of the link.

The deficiencies of such an approach soon became apparent. A series of telecommunications access methods (TAM's) were developed to perform many of the administrative functions

associated with line management. These provided the user with an interface to operating system functions which performed basic communications operations. An example of such a system is IBM's BTAM, which provided basic I/O operations on the line although the user was still responsible for the higher level functions of line management such as polling.

Later developments such as TCAM (2.1) provided a logical connection between the applications program and the terminal via a set of message queues, and a central message control program which performed the line-related input and output operations.

TCAM also provided a degree of independence between the program, which was regarded as a producer and consumer of messages, and the terminal itself by providing 'hooks' in the message control program where specific routines could be invoked to provide translations from device-dependent to device-independent formats. User programs could be written in a manner largely independent of data link and device control protocols, providing a general interface between terminals and applications.

VTAM (2.2) further enhanced general communicability between applications and terminals with the concept of a 'session' - a logical connection between two network-addressable units, referred to by symbolic names. Communications system routines mapped these identifiers onto physical device addresses, established the communications link, and added the necessary headers to each message to route it to its destination.

The above discussion has traced the development of communications software from its initial concern of merely providing an interface to the communications hardware to a

system which provides a logical connection between producers and consumers of information. The separation of different functions into distinct software modules within a hierarchical structure was a major factor in this development.

2.2.2 Physical Structure

The development of a functionally layered approach to communications software had a parallel in the development of hardware systems to provide increased performance and reliability of transmission-related functions.

The very nature of line control functions may require very frequent interruption of a processor when it is performing relatively simple tasks such as message assembly and polling. In some cases the overhead incurred in saving and restoring the processor environment status during interrupt processing may be of the same order as the useful work within the interrupt routine itself. Furthermore, the maximum latency time allowable in servicing the interrupt may typically be the assembly time of one character. Because of these requirements the first step towards distribution of hardware functions was to move the message assembly functions to a separate unit attached to the main processor. The increased capacity of this 'front-end processor' (FEP) enabled complete delegation of line management functions, and also motivated the development of more sophisticated data-link protocols to initiate, control, check and terminate data transfer over the link.

Techniques were also developed to improve the utilisation of communications lines by sharing a single line between many terminals, and to allow the efficient use of more cost-effective wide bandwidth lines. The primary techniques are:

- Multi-point (polled) operation

- Multiplexed operation

- Concentration

2.2.2.1 Multi-Point (Polled) Operation

In this method (sometimes called multi-drop) a line is shared sequentially between many stations on the line so that each station has full usage of the line for a limited time frame, normally one message. One station on the line, referred to as the primary, allocates the line to each station in turn by requesting input (known as a poll), or sending output. The data link control protocol includes a station address field in the message formats, and each secondary responds only to those messages addressed to it.

Multi-point operation provides an effective way of sharing a line between many similar terminals designed for the same data link protocol, particularly where the line capacity is several times greater than that required by a single terminal alone. It has the advantage that data is transferred to a station at the full line rate. However input operations incur some delay because the terminal must be polled before the data can be transmitted. Since there is no guaranteed allocation to each station, a busy terminal could lock out other terminals from accessing the line. The actual response time will be a probabilistic function of the line speed, message size, message rate per terminal and the number of terminals on the line, as well as the round-trip propagation delay for poll commands which may be particularly significant on half-duplex links. Various methods, generally based on some type of circular line loop in which the furthermost secondary is linked back to the primary, have been developed to increase the efficiency of the technique.

An alternative to the explicit allocation of transmission resources by polling is known as contention mode. This was first used in the Aloha Net (2.3) - A network based on the use of terrestial or satellite radio communications rather than fixed lines. In contention mode no attempt is made to coordinate the use of the transmission medium, but each station simply transmits data as it becomes available. This results in the possibility of a 'collision' in which two (or more) transmissions interfere with each other. When this occurs, the receiver will discard the complete message and each sender will retransmit the unacknowledged message after waiting for a random period. This method provides improved response time when the actual traffic rate is low in relation to the total channel capacity. The channel becomes saturated at 18% of its theoretical capacity - if all messages are of the same size and are synchronised to a common clock the saturation point is increased to 36% of channel capacity. More recently, this technique has been used in many 'local networks' with a high-speed coaxial cable rather than radio transmission.

2.2.2.2 Multiplexing

Multiplexing (together with the inverse operation of demultiplexing) refers to the combination of a number of communications channels onto one single channel of greater capacity. Each sub-channel is assigned some portion of the shared channel on a fixed predetermined basis. This apportionment may be achieved by two methods:

- Frequency-Division Multiplexing

  FDM splits the wide-band channel into a set of mutually exclusive subchannels of narrower bandwidth, each occupying a disjoint frequency range.

- Time-Division Multiplexing

  TDM allocates the entire channel capacity to each subchannel during a fixed time slot within a cyclic sequence.

Practical realisations of the time division technique can achieve more effective utilisation of the channel capacity. However some modems use FDM techniques to implement a low capacity 'back channel' to provide a limited simultaneous bidirectional link for control information. This allows a link which would otherwise be restricted to half-duplex (two way alternate) operation to avoid some of the inefficiencies introduced by line turn-round delays and operate at a higher aggregate data rate.

2.2.2.3 Asynchronous Multiplexing

In most cases a remote terminal will be actively transmitting or receiving data only a few percent of the time it is being used. A human operator typically interacts with a computer in a cycle of entering some request, waiting while it is processed, receiving the response, and thinking before repeating the cycle. Therefore in any system with a static allocation of transmission facilities (such as synchronous multiplexing) the channel is still inefficiently used.

Asynchronous Time Division Multiplexing (ATDM) or 'statistical multiplexing' is one technique which has been developed to overcome the inherent inefficiency of static channel allocation. The development of the microprocessor based 'statistical multiplexor' has enabled much higher channel utilisation to be achieved merely by the substitution of an 'intelligent black box' in place of a normal multiplexor, although it offers no greater functionality. With ATDM each

data unit (normally one character) is preceded by an address specifying the sub-channel to which it belongs, and any sub-channel has the opportunity of using the next available time slot on the link rather than being restricted to a particular place in the overall sequence.

2.2.2.4 Concentration

The techniques examined so far fall into two categories.

Polling sequentially allocates the total capacity of the line to a station for the complete duration of a message through the handshaking implemented within the data link protocol. The actual data transfer utilises the full capacity of the transmission link, and allocation overhead is incurred only once per message. The data link protocol provides some degree of message integrity checking, but requires each station to implement the data link control protocol and provide local message buffering. The input polling may introduce considerable overheads.

Multiplexing provides quasi-parallel operation for each sub-channel by mapping several lower capacity sub-channels onto one higher capacity physical link in such a way that each sub-channel appears indistinguishable from a dedicated link. Because multiplexing affects only the physical link between end-points it is transparent and has no effect on the higher levels of the communications system. Multiplexing does involve some inefficiency due either to the static sub-channel allocation resulting in unused time slots or to the address information which must be attached to each character transmitted.

Concentration attempts to combine the advantages of both the multipoint and multiplex techniques while avoiding the

disadvantages of either, although this requires a much higher degree of intelligence in the line sharing unit (concentrator).

Because many stations may transmit at the same time the concentrator must provide buffering of input until time becomes available on the line to transmit it. Likewise data may arrive at a greater rate than the receiving station can accept it, so further buffering may be required. In practice some limit must be placed on the size of transmitted messages together with some form of pacing or flow control to prevent saturation of the conentrator by any one station. The amount of buffering and the complexity of buffer management may vary considerably depending on the diversity of terminal types, line speeds and message lengths.

Concentration enables much fuller utilisation of the communication link capacity to be obtained since buffering has a smoothing influence on traffic flow, providing a degree of isolation between the traffic on the communications line and each individual station. The use of intelligent communications controllers at both ends of the line also enables the use of more sophisticated communications protocols.

2.2.3 Summary

Terminal access networks have evolved in two distinct areas; device independence and line control procedures.

A considerable degree of device independence and data link protocol independence was achieved by inserting intermediary data formatting levels (known as presentation services) between the user program and the communications line protocol handlers. In general, user programs had no need to be concerned

with the nature of the line protocol or details of the device control codes for particular terminals.

Likewise with the advent of intelligent concentrators the management and sharing of communications line resources became a separate function from the particular device using the communications facility. This led to the development of standardised data link control protocols implemented by a wide range of devices.

However the onus for adapting to the characteristics of any particular device remained with the user of that device - hence if one device could be accessed from many systems then each system would be required to implement data formatting services for that device rather than the device itself adapting to some common standard.

2.3 MULTI-COMPUTER NETWORKS

The preceding discussion of terminal access networks covered what are commonly known as 'star' networks involving the connection of many remote terminals to a single central computer complex. As functions were distributed to different parts of the network, the basic star structure evolved into more complex hierarchical structures, but the one-to-one relationship between terminals and hosts remained.

When multi-computer networks are considered, such a simple relationship between terminals and hosts no longer applies. A terminal may wish to communicate with one host, or alternately with many hosts; hosts may wish to communicate with other hosts or terminals may wish to communicate with other terminals. In fact the distinction between hosts and terminals may largely disappear, with both being regarded

simply as end users of the communications system.

2.3.1 Packet Communications Systems

The efficient utilisation of the transmission link capacity has been a major consideration in the development of the various line sharing methods. The most efficient technique developed, concentration, is characterised by two key attributes:

- Data is buffered within the concentrator, which has a smoothing effect on the data flow, allowing a higher average throughput on a line of given capacity.

- Terminal multiplexing or polling is performed locally by the concentrator. The data link protocol between the host and the concentrator may be designed to maximise the utilisation of the link capacity.

Packet switching is an extended form of concentration, just as concentration itself is an extended form of ATDM. Concentration multiplexes messages between end points which are physically connected by a single communications link. In the process of packet switching however, the source and destination first agree to a logical connection. The logical connection between source and destination may be implemented by more than one physical circuit, in parallel or in series.

Messages may be segmented (if necessary) into smaller portions of some fixed maximum size and each segment transmitted individually across the network to the destination, where they will be reassembled back into a complete message.

At each intermediate point routing tables determine the next link in the journey to the destination, and when dynamic routing algorithms are employed successive segements may take different routes across the network and even arrive out of sequence.

Because message flow can be controlled between the source and the destination, and the network data units are fixed in size, buffering and queue management within the network can be simplified. This minimises transportation delays within the network and results in a high rate of data throughput.

Packet oriented communications networks are the preferred technique for future communications systems, because

- Packet networks provide high throughput and fast response times due to the efficient utilisation and sharing of the communications facilities.

- Packet networks provide a general communicability since a transparent logical connection can be established between any two termination points of the network independently of the actual physical paths involved.

In addition, international standards have been developed from the outset which define packet interfaces to Public Data Networks (PDNs), providing the facility of national and even global data communications in a similar manner to the voice telephone networks of the present day.

2.3.2 Classification of Network Architecture

Since in a multi-computer network the logical connection between end users may be established dynamically (rather than statically as determined by the physical hierarchy), a resource manager must be provided to initiate and terminate logical connections. Based on the distribution of this resource management function, two classes of configurations may be defined:

- Host establishment of session (Fig. 2.1-A).

  If a terminal is associated with a local network of a 'parent host' the establishment of a logical connection with another host requires the parent host to arrange the connection. Once this is done traffic may either pass through the parent on its way to the destination, or pass directly from the origin to the destination. Two terminals wishing to communicate can only do so through a common host.

- Network establishment of session (Fig. 2.1-B).

  If both terminals and hosts are regarded simply as addressable units on the network the establishment of a logical connection between a terminal and a host or another terminal is purely an internal network function. Within the network terminals will most likely be associated with the local network of a network nodal processor, but this is transparent from the user viewpoint.

2.3.3 Effects on Network Architecture

The 'parent host' approach is typified by the Systems Network Architecture (SNA) of IBM (2.4). The SNA software was

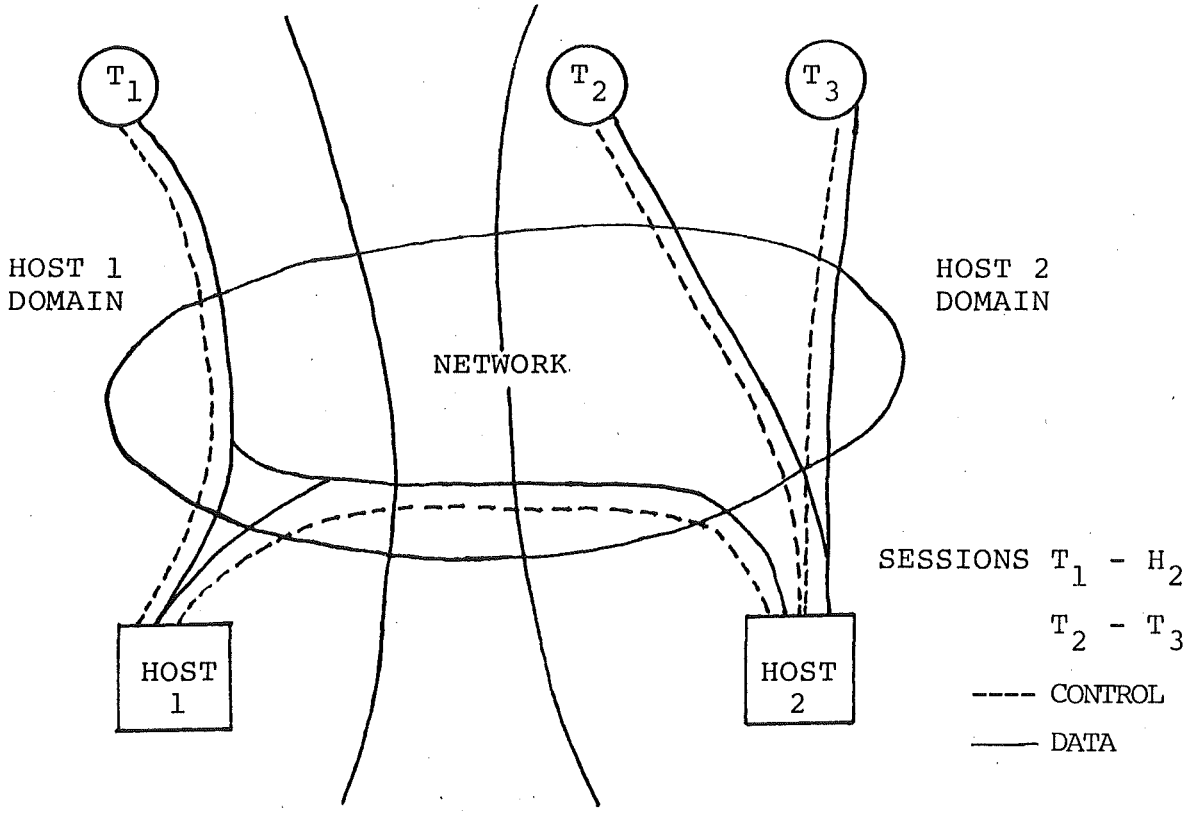


FIG. 2.1-A HOST ESTABLISHMENT OF SESSION

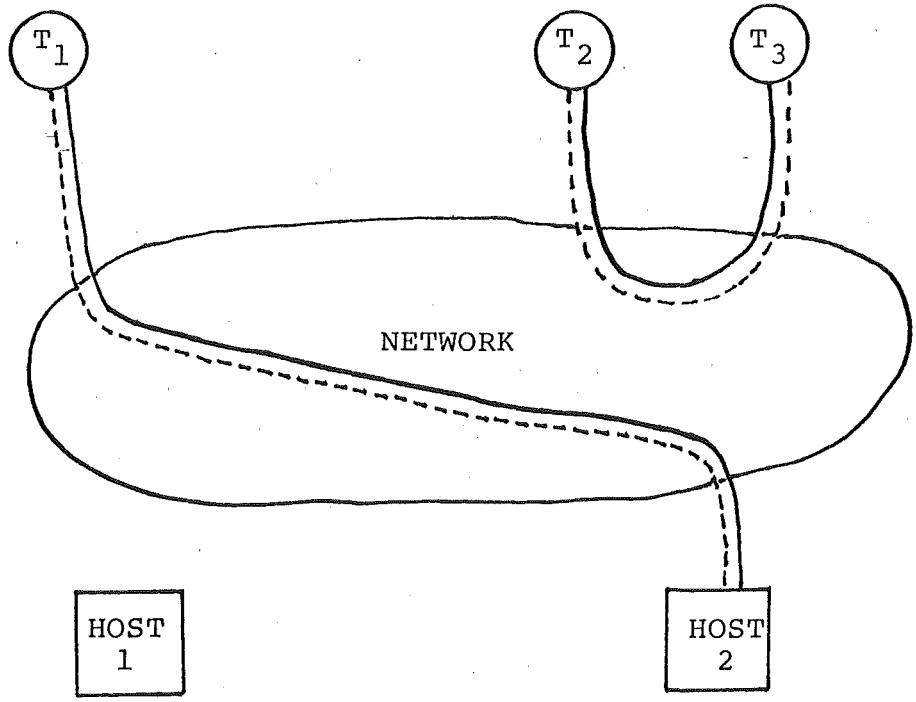


FIG. 2.1-B NETWORK ESTABLISHMENT OF SESSION

initially developed to unite the communications components of existing communications access methods (eg. TCAM and VTAM) and applications subsystems (eg. IMS, CICS and JES) under a single umbrella in order to provide a migration path towards an integrated communications architecture. SNA was implemented to run on the same SYSTEM/370 3705 communications controller as the earlier software, thus being heavily dependent on the host SYSTEM/370 for network control functions. More recent developments have enabled coupling of SNA networks running on multiple hosts and the connection of smaller machines, but the orientation towards a dominant 'host' controlling the network remains.

SPERRY UNIVAC's Distributed Communications Architecture (DCA) (2.5) provides a contrast in that network operation is autonomous of any host computer that may be connected. All devices connected to the network are co-equal in the sense that each is regarded simply as a producer and consumer of messages. The lower Transport Network (TN) level provides a data transmission facility between any pair of network ports, while the higher Communications System User (CSU) level provides a network interface of the appropriate functionality for the particular device.

2.4 SUMMARY

The one attribute which characterises the development of communications systems must surely be diversity. Diversity is everywhere; in the lower levels of device control protocols, through data link protocols and transmission techniques, to the highest levels of network management. To a large extent this is due to each successive generation of communications products having evolved from its predecessor in a reaction to the

requirements of users rather than as a stage of a planned development.

The examples considered here have been mainly from large-system data communications networks. They remain valid when considering the interconnection of smaller systems, since it should be possible both to avoid known shortcomings of earlier approaches, and to determine from the outset those areas requiring particular attention.

3. COMPUTER NETWORK STRUCTURE AND IMPLEMENTATION

This chapter surveys the current development of computer networks. After an initial introduction to data link protocols, the major paths of development are discussed, with particular emphasis on local computer networks. Finally two alternative approaches to the implementation of network software are examined.

3.1 DATA LINK CONTROL PROTOCOLS

A data link control protocol has three essential functions, being:

- *Framing*

  The process of recognising the start and end bit sequences which delimit messages. These may not only bracket the actual information transmitted, but also indicate portions of the message to which the checking mechanism applies.

- *Link Management*

  The process of controlling transmission and reception on the link. This may include the direction of traffic on a half-duplex link, the particular station which may use the link, identification (addressing) of receiver and sender, and link set-up. Possible link configurations may include point to point (dedicated link between two stations), multi-point (polled operation) and switched line (dial-in/ dial-out).

- *Data Transfer and Message Integrity*

  The process of transferring data sequentially and without error over the link. This may include regulating the rate of data transfer (flow control). Errors may be detected by parity checks (vertical and horizontal) or cyclic redundancy codes, or by message sequence numbers. Flow control and recovery may be implemented by two basic methods:

  - A *'Stop-Start'* method in which each message requires an explicit acknowledgement and go-ahead response before the next message is transmitted.

  - A *'Continuous Modulo N'* method in which frames are sequence-numbered and may be transmitted continuously until a 'credit limit' on the number of outstanding unacknowledged messages is reached, when a response is specifically requested.

A highly desirable attribute is:

- *Transparency*

  It is often necessary to transfer binary data which may contain arbitrary bit patterns rather than characters from some restricted set. Most protocols provide some mechanism for the transfer of such data while maintaining the ability to distinguish control information from data.

The communications protocols in common use may be divided into two basic categories; byte control protocols (BCP's) and the more recently developed bit-oriented protocols (BOP's). BSC (BISYNC) and the SDLC/HDLC-based family are the most well-known examples of each type respectively, while DDCMP has characteristics of both. In all cases the basic components of a message are the header of control information, user data, and error-detection codes.

3.1.1 Byte Control Protocols

In byte control protocols, characters from a reserved subset of the character set (known as control characters) are used to frame the message, delimit message components, and perform link management functions.

In BISYNC the start of the variable length header is indicated by the SOH (Start of Header) character. The header is terminated and the start of the data portion of the message is indicated by the STX (Start of Text) character. The data may be termined by an ETX (End of Text) character, or an ETB (End of Text Block) when the message has been segmented and more blocks are to follow.

Line control commands for polling and addressing, and their associated responses, are implemented by separate control messages. These consist of a one or two character control sequence, rather than being contained within a standard-format header. Data security is implemented by a checksum over the data portion of the block, but since this is not applied consistently over the entire frame, addresses and control sequences are not protected.

The problem of control characters in the user data being interpreted incorrectly is overcome by a special feature known as *transparent mode*. If the text is to be sent in transparent mode, it is bracketed by DLE STX and DLE ETX (or DLE ETB). The DLE indicates that the following text may contain embedded control characters. If a DLE character appears in the text, a second DLE is inserted by the sending station. In turn, the receiving station discards one DLE of each consecutive pair. This process is known as *byte-stuffing*.

The DDCMP protocol uses a frame structure for all data messages. Each message begins with one control character to distinguish between data, control and bootstrap frames. The header contains a count field specifying the length of the data, as well as sequence and acknowledgement information. The use of a count in this manner achieves data transparency without the use of byte-stuffing. Routing and addressing functions are performed by a separate Network Services Protocol (NSP) format header preceding the data frame.

3.1.2 Bit-Oriented Protocols

An alternative to using a reserved control character set, with the inherent problem of these characters appearing in the data, is to rely on *positional significance* within a structured control field. A structure is positionally significant when the semantics of fields within it are defined by the relative bit positions, rather than character content.

The HDLC/ADCCP protocol (standardised by ISO and ANSI respectively), together with variations implemented by the various computer manufacturers, form the family of bit-oriented protocols. They have two distinguishing

characteristics, the frame format and the transparency technique.

There is a single frame format (or communications envelope) for all message types and link configurations, in comparison to the numerous message formats used by the byte-oriented protocols. The frame is delimited at the beginning and end by a special binary sequence, 01111110, which is called the *flag*. The header follows the initial flag, consisting of an 8-bit address field, and an 8-bit control field which may be interpreted in three formats depending on the setting of the first two bits. The 16-bit cyclic redundancy code, which immediately precedes the trailing flag, checks the entire frame.

Transparency is achieved by the technique of zero insertion and deletion, more commonly known as *bit-stuffing*. After transmitting the initial flag, the sending station monitors the output bit by bit, rather than character by character as in byte-oriented protocols. Any time a string of five ones occurs, the sending station will automatically insert an extra zero into the transmitted stream to avoid the possibility of a flag sequence being transmitted unintentionally.

The receiving station monitors the incoming bit-stream. When five consecutive ones are detected, the following bit is examined. If it is a zero, the bit has been 'stuffed' by the transmitter and it is discarded, rather than being included in the message being assembled. If the sixth bit is a one, the sequence may be a flag (the next bit being 0) or one of two additional control sequences. These are the *go-ahead* (7 ones) used in loop configurations, and the *abort* (8 to 15 ones) used to abort a partially transmitted frame. Since both these sequences cannot occur within a normal frame, there can be no ambiguity.

3.2 DEVELOPMENT OF NETWORK TECHNOLOGY

Communications network technology has evolved along three paths, oriented towards the particular applications requirements of the developers; the computer manufacturers, the telecommunications industry, and private or academic institutions.

3.2.1 Computer Manufacturers

Among the computer manufacturers, DECNET (3.1), SNA (3.2) and TELCON (DCA) (3.3) are the most mature products available. They are characterised by the provision of higher level services tailored towards the particular applications environment. SNA and TELCON, oriented towards the commercial data-processing environment, provide 'presentation services' which perform mappings from a network-wide device-independent (virtual terminal) protocol to the particular device control protocol for a variety of terminals. DECNET is designed for the interconnection of minicomputers, which may be running under different operating systems. With an orientation towards distributed and real-time processing, it provides a File Transfer Protocol (FTP) to facilitate the cross-generation of software and remote file access between machines.

3.2.2 The Telecommunications Industry

The telecommunications industry has traditionally provided analogue (voice) communications between subscribers connected via local exchanges and the toll trunk network. The rapid development of electronic technology and the emergence of distributed data processing systems has resulted in a growing need for digital data transmission facilities. While the volume of voice traffic is many times greater than the current

and forseeable data traffic volume, the introduction of new technology provides an opportunity to establish from the outset common interfaces that will allow the direct interconnection of public data networks on a global basis.

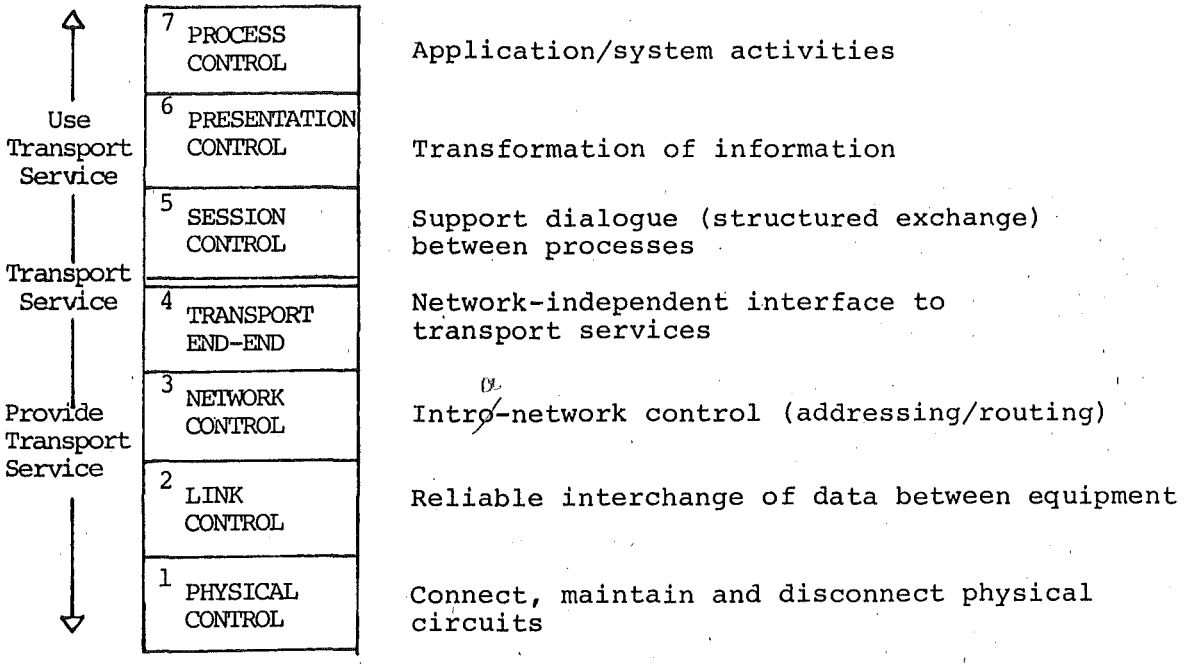
The trend towards standardisation has taken two directions. The international standards organisation, ISO, has developed a common reference model defining terminology and functions to facilitate the analysis and interconnection of different networks. The international telecommunications union, the CCITT, has developed the X.25 Recommendation to provide a standard interface to public packet-switching data networks.

3.2.2.1 Open Systems Interconnection

Open Systems Interconnection refers to the transparent interconnection of different computer systems at all levels from the physical communication medium through to the application system.

ISO has developed a reference model (3.4) for open systems interconnection which defines the structure of a communications system in terms of a seven-layer model (Fig. 3.1). The purpose of the model is to provide a complete reference structure for communications systems, partitioned into layers of functionally contained components which are each easily comprehended. The model is intended to:

- Provide a common perspective from which to evaluate existing systems
- Identify those areas requiring further development and standardisation
- Allow easy mapping onto potential practical implementations



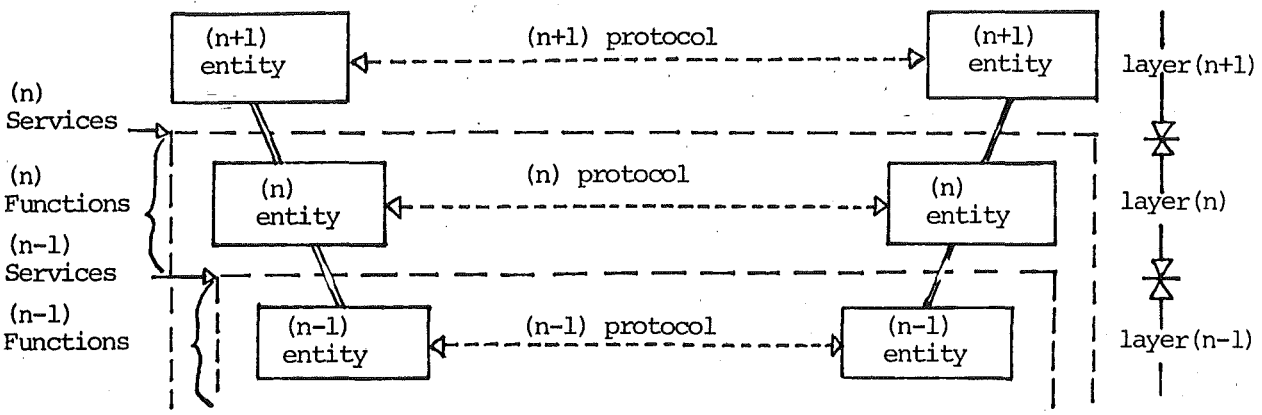


FIG. 3.1 ISO REFERENCE MODEL

The layers comprising the model are:

- *Process Control* (Level 7)

  Performs activities to support the information processing function of the application. Examples are remote batch, file transfer, or terminal interaction protocols.

- *Presentation Control* (Level 6)

  Provides the required transformations to information being transferred. Examples are encryption, data compression, and terminal virtualisation.

- *Session Control* (Level 5)

  Supports a dialog (structured exchange) between users. This includes resolving symbolic user addresses, the establishment and termination of sessions, and the structured interchange of data between users. It uses the transport network to exchange messages between nodes.

- *Transport End-to-End Control* (Level 4)

  Provides the interface to the transport network, implementing end-to-end control and information exchange across simple or complex networks.

- *Network Control* (Level 3)

  Provides the control functions of switching points for intra-network operation, such as addressing and routing.

- *Data Link* (Level 2)

  Provides the reliable interchange of data between equipments connected by Level 1 facilities.

- *Physical Link* (Level 1)

  Includes the physical, electrical, functional and procedural characteristics of the physical circuits between equipments.

3.2.2.2 The X.25 Public Data Network Interface

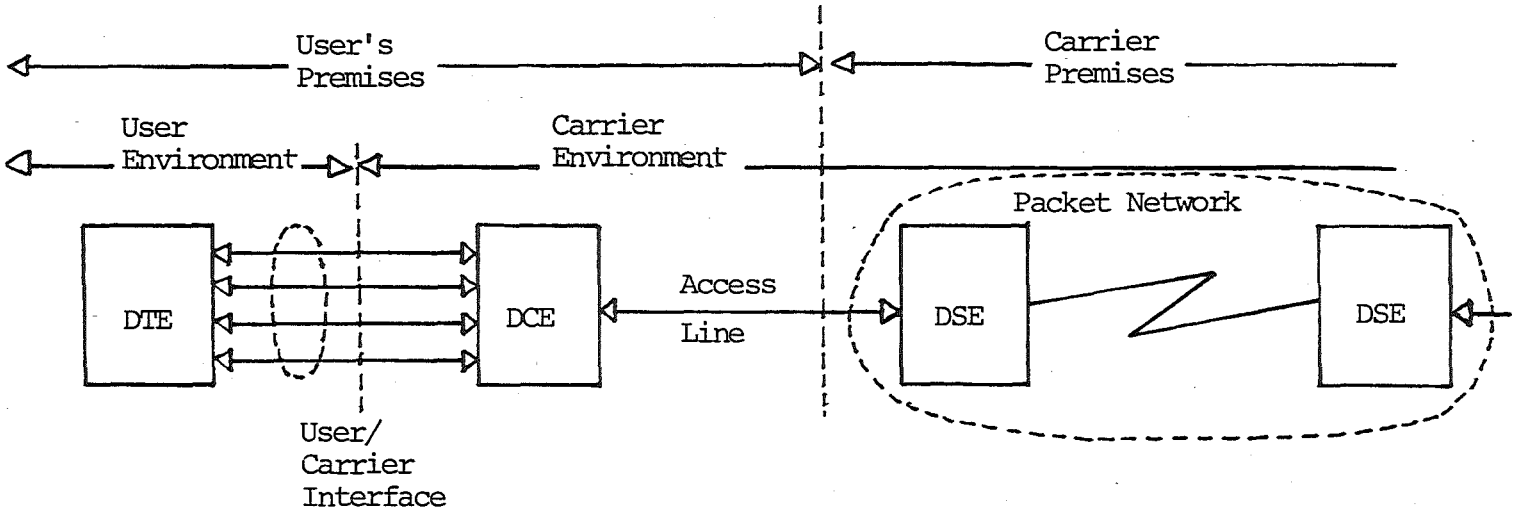
The development of the X.25 interface (3.4) has been very significant. A brief description follows, using the terminology of the telecommunications industry as illustrated in Fig. 3.2.

The Data Terminal Equipment (DTE) can be any type of user facility from a simple terminal to a large computer system. The Data Circuit-terminating Equipment (DCE) performs the interface function into the network proper, which is known as the Data Switching Exchange (DSE).

The CCITT has standardised the protocols across the DTE/DCE interface. This interface is considered at three levels. These are:

- *The Physical Level*

  The electrical interface between the DTE and the DCE is specified by the CCITT X.21 interface, or the X.21 BIS interface which is retained for compatibility with older implementations.



DTE Data Terminal Equipment

DCE Data Circuit-Terminating Equipment

DSE Data Switching Exchange

FIG. 3.2 CCITT TERMINOLOGY

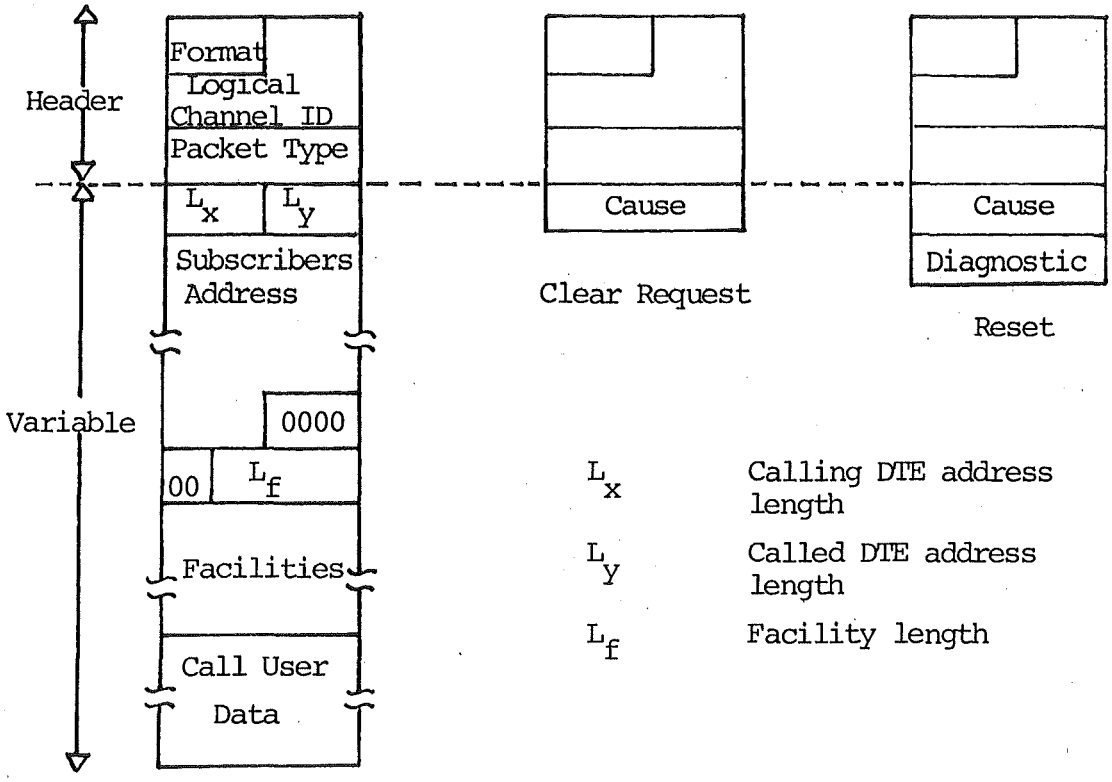
- The *Link Access Procedure* (LAP)

  The bit-oriented LAP and LAP-B data link control protocols provide a transparent bi-directional transfer of data blocks between a station in the DTE and a station in the DCE. The earlier LAP implemented an independent link in each direction using the HDLC Asynchronous Response Mode (ARM), while the more recent (and preferred) LAP-B uses the full-duplex Asynchronous Balanced Mode (ABM) or HDLC.

- *The Packet Level*

  The packet level defines a standard packet header and formats for the construction of data and control packets (see Fig. 3.3), and procedures to establish, multiplex and control the flow of data over many virtual calls on the single physical link between the DTE and the DCE.

It must be noted that the X.25 Recommendation provides a mechanism only for the transparent transfer of data blocks between paired DTE's across a public data network. Since X.25 is an access method, it may be related to different levels of the ISO model, depending on the way the X.25 link is used. If each terminal uses a unique X.25 call then the X.25 network performs the transport level functions directly. If, however, the number of virtual calls that can be supported is restricted then a separate transport level is required to perform end-to-end multiplexing of multiple terminals onto a single X.25 call at the network level. Higher level user-to-user protocols are beyond the scope of the X.25 interface. The latest draft Recommendation (3.6) has deleted references to protocol level numbers to avoid conflict with the ISO model.



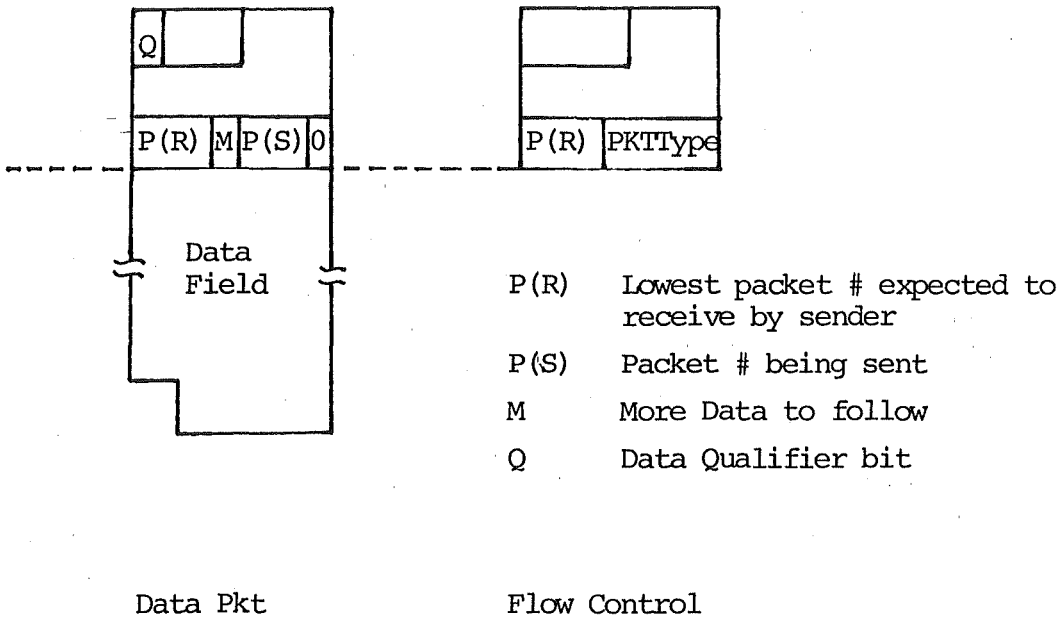


FIG. 3.3 SAMPLE X.25 PACKET FORMATS

Among networks currently using the X.25 interface are the American TELENET, the French TRANSPAC and the Canadian DATAPAC networks.

3.2.3 Private (Local) Networks

Most academic or research networks fall into the class of what are becoming known as *Local Computer Networks* (LCN's). A variety of definitions have been proposed for local networking (3.7), but none has found general acceptance. In this discussion local networks will be characterised by the following attributes:

- Owned by and operating within a single organisation
- Short distances (under 10 km)
- High data rates
- Some form of mesh (packet switching) or broadcast (data bus) structure rather than the conventional star (hierarchical) network structure.

The usage of LCN's have evolved in two distinct applications:

- The improvement of existing systems by the addition of 'frontend' communications systems or 'backend' peripheral systems. The Ethernet (3.8) is possibly the most well-known example of a high-speed broadcast packet communications network, while the HYPERCHANNEL (TM) (3.9) is a high-speed bus-oriented network designed to facilitate backend communications between large mainframe and peripheral systems.

- The exploration of new systems concepts such as distributed processing. CM* (3.10) and C.MMP (3.10) are two well-known distributed multi-processor systems utilising a

** HYPERCHANNEL is a trademark of the Network Systems Corporation.

high-speed packet switch and an interprocessor bus network respectively.

The actual application of local network technology is limited. Many systems remain laboratory curiosities. At the time of writing, the HYPERCHANNEL (TM) is the only system commercially available, but Xerox, Digital Equipment Corporation (DEC) and Intel are jointly developing a common Ethernet standard.

3.2.4 Summary

Communications network technology has developed in different directions in response to differing market needs.

The computer manufacturers have tailored their network systems towards their particular market segments with a heavy emphasis on end-user facilities and the support and integration of existing products within a new framework.

The telecommunications industry is primarily concerned with the control of access to public data networks.

The various local networks have been developed as pragmatic solutions to the requirements of particular organisations. Free from the restrictions of ancestral heritage and the need to function in an arbitrary user environment they often display a stark simplicity of design.

3.3 LOCAL COMPUTER NETWORKS

This section examines the structure of two local network implementations. The particular two have been chosen because they demonstrate a wide variation of architecture while

meeting similar objectives; CNET uses broadcast transmission of large (up to 256 byte) packets while MININET performs store-and-forward switching of 16-bit data words.

3.3.1 CNET

CNET was developed in the Computer Systems Laboratory, Queen Mary College (University of London). The network architecture is described in (3.11) while (3.12) contains implementation-level detail.

3.3.1.1 Evolution Context

CNET is an inexpensive packet communications network for the interconnection of many devices, which may vary in intelligence from printers and terminals to computers. It evolved from the requirement for a flexible interconnection medium which would be extensible, reasonably low cost, yet powerful, in order to construct distributed systems as well as provide a conventional node-to-node communications system. The design is heavily influenced by Ethernet.

3.3.1.2 Data Transport Technology

Node controllers are connected via a common passive coaxial cable referred to as the *ether*. Packets consist of an 8-byte header, up to 256 bytes of user data and a 1-byte checksum, with transparency being achieved by a data count in the header. Packets are broadcast across the ether, and each data packet is individually acknowledged.

A host device wishing to send a message passes the message to its associated controller. The controller monitors the ether and, if the ether is in use, waits until the current

transmission has completed. When the ether is clear, the controller immediately begins to transmit the queued packet. This packet is received by every controller on the ether but ignored by all but the addressed destination.

If more than one controller attempts to transmit simultaneously, a collision will occur resulting in packet corruption. Each transmitting controller performs *collision detection* by simultaneously receiving the data on the ether as it is transmitted. When a mismatch is detected, indicating a collision or noise corruption, the sending controller immediately ceases transmission and forces the ether into a 'break' state, which causes all receiving controllers to discard the packet. After a random timeout the controller will attempt to retransmit the packet.

The destination controller performs a parity check before passing the packet to its attached device and returning an acknowledgement to the source. A node may have only one packet outstanding to each other node, which must be acknowledged before another packet can be sent to that node.

3.3.1.3 Node Structure

A minimal CNET controller consists of a Motorola M6800 microprocessor, a 10 msec real-time clock, 2K bytes program memory, 4K bytes buffer memory, an Asynchronous Communications Interface Adaptor (ACIA) to the ether and another ACIA or parallel port to the device. The approximate cost of the hardware is $180.

The structure of the controller software is summarised in Fig. 3.4-A and the packet format in Fig. 3.4-B. Data may be transferred simultaneously to and from the host under interrupt control, with each packet requiring acknowledgement.

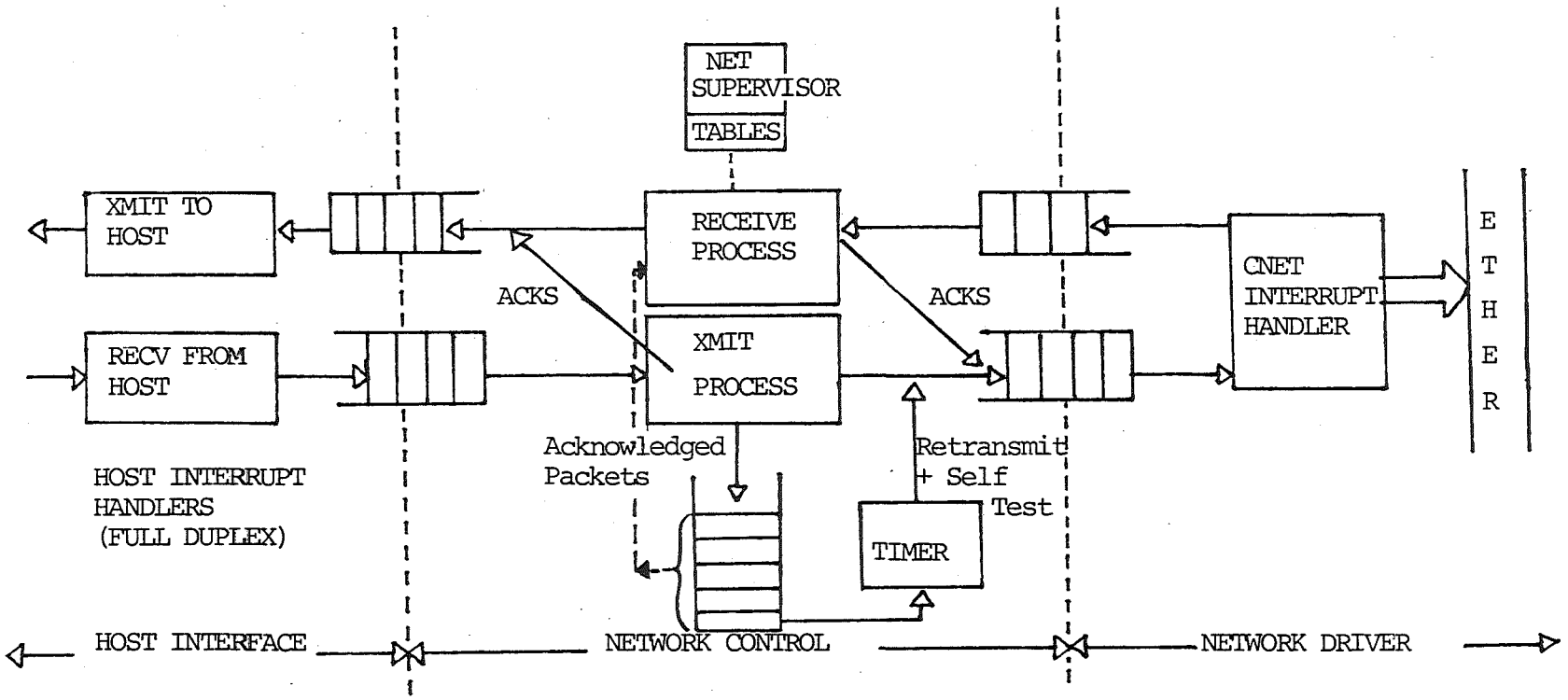


FIG. 3.4-A CNET CONTROLLER STRUCTURE

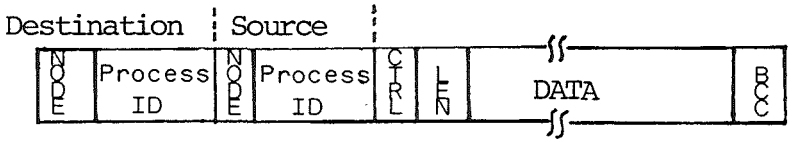


FIG. 3.4-B CNET PACKET FORMAT

Since reception and transmission are alternate rather than simultaneous, a single interrupt handler performs both operations. At the completion of receiving any packet the interrupt handler will attempt to initiate any pending transmission. During transmission the actual data on the ether is monitored through the receiver interface to detect 'collisions' or other data corruption.

At the packet level independent transmit and receive processes monitor their respective input queues transferring data packets to their output queues, generating and processing acknowledgement packets, and interpreting the network control packets which establish and terminate logical network connections.

A timer driven from the real time clock performs collision resolution timing, transfers packets from the 'acknowledgement pending' queue to the transmit queue for retransmission when they time out, and periodically schedules self-test packets.

3.3.1.4 Network Interface

The physical interface between the controller and the device may be either an asynchronous character interface for terminal-like devices, or a parallel port connecting to a minicomputer Direct Memory Access (DMA) controller.

Logically the controller appears as a 'door' between the host and the ether, simply transferring packets between addressed nodes. A CNET controller could conceivably contain alternative interfaces implementing a virtual terminal protocol or a 'gateway' transformer to another network protocol, but the only interface described in the references is to PDP-11 UNIX systems.

3.3.1.5 Connection Management

As indicated in Fig. 3.4-B packet addresses consist of two fields - an 8-bit node number and a 16-bit port or process identifier identifying the logical channel within the host. The controller administers a table of open inter-port connections consisting of (source, destination) tuples, which is updated in response to control messages originating from a network supervisor process residing in the host connected to the controller. The function of network supervision is thus split between the controller and the host.

3.3.2 MININET

MININET is a joint development of the Polytechnic of Central London and the University of Bologna. Reference (3.13) describes the packet protocol in detail, while (3.14) gives an overview of the network concepts and a description of the implementation.

3.3.2.1 Evolution Context

MININET is a packet switching data communications network developed to facilitate the interconnection of heterogeneous locally dispersed digital devices. These devices may not only be minicomputers and terminals, but also laboratory instrumentation which has a digital interface.

3.3.2.2 Data Transport Technology

Architecturally MININET consists of two types of nodes; the Stations which provide the external interface to the network and the Exchanges which implement a store-forward packet

switching data transport service interconnecting the controllers.

The Stations act as local concentrators and provide the interface to connect the users to the network. A Station may provide up to 64 ports (physical access points). The Station structure is illustrated in Fig. 3.5.-A.

The Exchanges route packets towards the destination station, according to addresses contained within the packets. Up to 64 Stations and Exchanges may be connected together in a completely flexible configuration. The Exchanges are structurally similar to the Stations, but consist of multiple channel sections interconnected by the central switch.

The packet structure, illustrated in Fig. 3.5-B, consists of 16 bits of user data and 16 bits of control information embedded within a 44 bit communications frame. The MININET link protocol is based on a one-to-one correspondence between packets transmitted and received over a link, with a data packet in one direction carrying 'piggyback' the acknowledgement for the previous packet in the other direction. If necessary 'dummy' packets are transmitted to maintain a balanced flow.

Error recovery makes use of a modulo-8 sequence number within the communications frame. Normal incrementing is interpreted as an acknowledgement of the last packet transmitted; a break in the sequence indicates an error situtation and causes retransmission beginning at the first unacknowledged packet.

3.3.2.3 Node Structure

Three different node implementations which have been developed are:

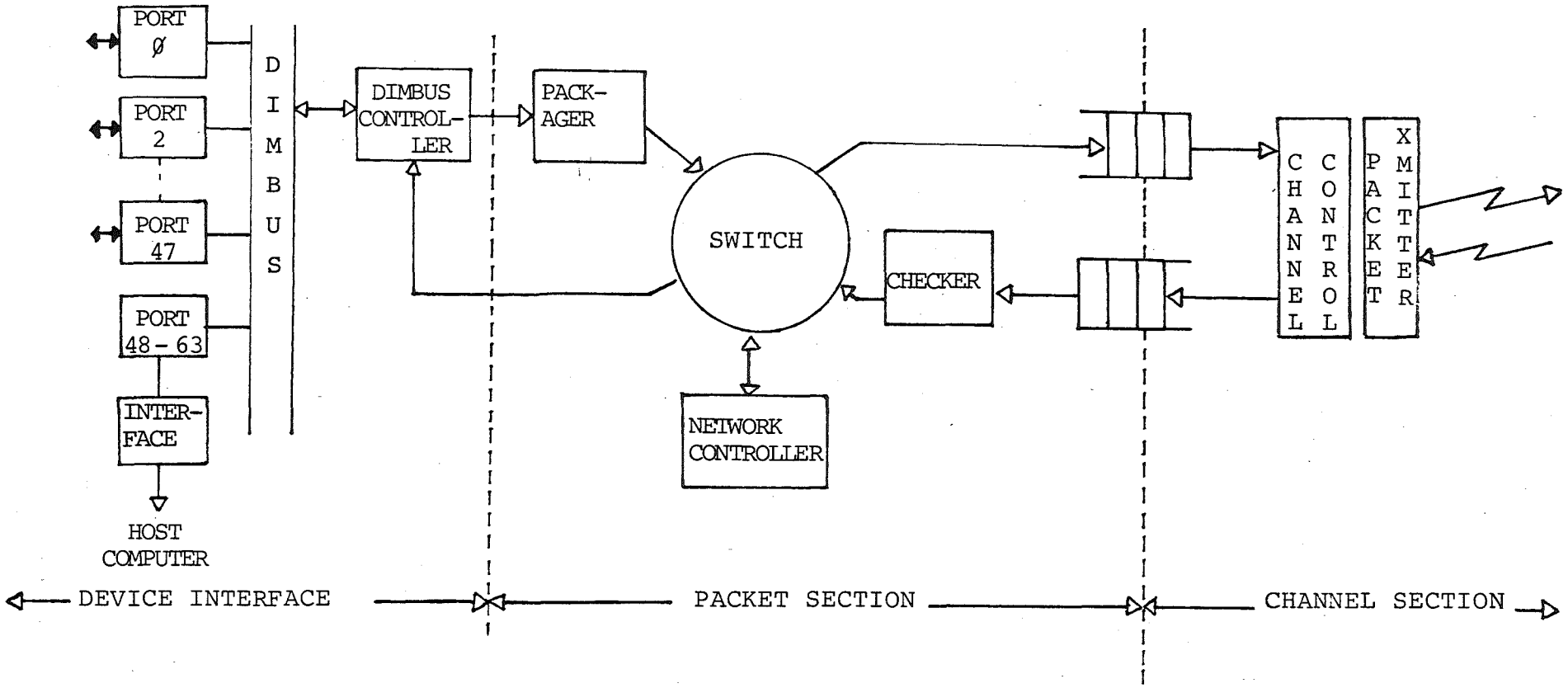


FIG. 3.5-A MININET STATION STRUCTURE

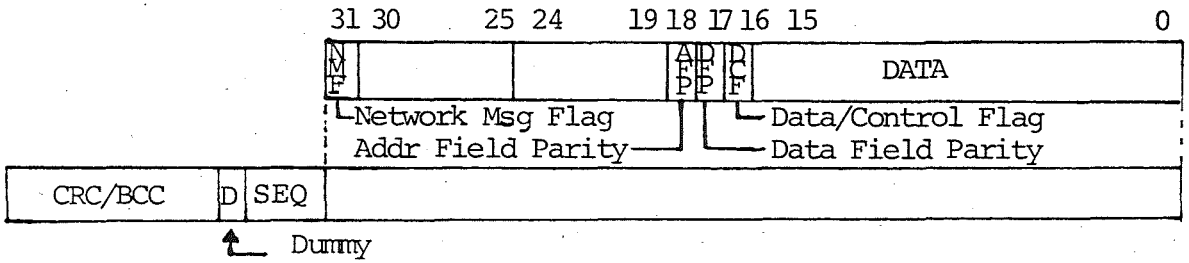


FIG. 3.5-B MININET PACKET PROTOCOL

- The slow speed Station, handling about 100 packets/sec, uses a 16-bit National Semiconductor PACE microprocessor, a real-time clock, parity generation and checking hardware, and a Universal Asynchronous Receiver/ Transmitter (UART) to interface to the data link.

- The medium speed Station, with a throughput of about 1000 packets/sec, uses a second PACE microprocessor dedicated to handling the data link functions and a special communications hardware board.

- The high speed Station, with a throughput of up to 100K packets/sec, uses microprogrammed hardware based on the AMD 2909 sequencer to implement the packet transmission facilities, with the PACE microprocessor performing network management functions only.

The packet section implements the logical functions required to transmit data between a pair of ports. Outgoing data passes through the *packager* which creates a packet by the addition of control bits and the destination address, which are processed by the *checker* on input. All packets pass through the *switch* which performs internal routing, steering packets from the packager and the checker to the DIMBUS controller, the network control section or the output queue depending on the NMF flag and the address bits.

The channel section implements the MININET data link control protocol. It consists of an interrupt handler performing packet framing and synchronisation, and a higher level process which implements the frame-level protocol. This process, which is activated each time a packet is received, checks the frame fields for validity before placing the packet on the input queue and initiating the transmission of the next output packet, or a dummy if the output queue is empty.

3.3.2.4 Network Interface

Each device connected to MININET must be provided with a simple hardware interface to the DIMBUS, a simple full-duplex data bus with a centralised controller. This interface may be a single port interface for a simple device, or an input/output controller which enables a minicomputer to access 16 ports through one physical interface. Each port provides a 16-bit data interface and a control bit to distinguish between user data and network control information.

3.3.2.5 Connection Management

A connection between two ports is established by using the DCF flag to indicate control mode and specifying the destination node and port. The network control sections of the source and destination nodes establish a logical connection using control packets, and then data transfer may proceed.

3.3.3 Summary

Packet communications technology, as discussed in this section, is well developed. A wide variation in architecture is possible, including packet-switched, broadcast and bus technology. The two local network systems considered (CNET and MININET) display a wide variation in switching technology (broadcast vs. packet switch) and in packet structure (variable up to 256 bytes data vs. fixed 16 bits data). Provided that the packet structure is appropriate to the applications environment, it appears that both conventional packet switching and broadcast techniques are capable of providing adequate performance at low cost. Many local networks employ broadcast techniques because packets can be transmitted directly to the destination without the added

complication of intermediate buffering and routing, and the ease and flexibility of altering the configuration.

3.4 NETWORK IMPLEMENTATION TECHNIQUES

This section examines two distinctly different approaches to computer network implementation.

3.4.1 Low Level (Configurative) Approach

The control software for most communications systems is implemented in the assembly language of the particular communications processor. A specially written device handler routine is provided to interface specific types of terminals to the network - such as teletype-like devices, the IBM 2741 family of polled terminals using the BISYNC protocol, or Remote Job Entry (RJE) stations.

The process of generating control software for a specific physical network is known as *configuration*. The user specifies the physical structure of the network in terms of the lines and stations, and the terminal types from a predefined set of supported devices (Fig. 3.6). The configuration statements are generally interpreted by a macro-processor to produce a set of tables which define the structure and relationships of network entities, and a set of flags which are used to control the conditional inclusion of code during the communications control program assembly.

This approach is generally limited to a selection between a set of predefined options. The user who wishes to support a 'foreign' terminal must write a terminal handler in assembly

```
CLUST078    CLUSTER     CUTYPE=3271,GPOLL=407F,LINE=078
            TERMINAL    TERM=3277,SELECT=6040,FEATURE=OPRDR,MODEL=2
            TERMINAL    TERM=3277,SELECT=60C1,FEATURE=OPRDR,MODEL=2
            TERMINAL    TERM=3277,SELECT=60C2,FEATURE=OPRDR,MODEL=2
            TERMINAL    TERM=3277,SELECT=60C3,FEATURE=OPRDR,MODEL=2
            TERMINAL    TERM=3277,SELECT=60C4,FEATURE=OPRDR,MODEL=2
            TERMINAL    TERM=3277,SELECT=60C5,FEATURE=OPRDR,MODEL=2
            TERMINAL    TERM=3286,SELECT=60C6,MODEL=2
            TERMINAL    TERM=3284,SELECT=60C7,MODEL=2
CLUST07A    CLUSTER     CUTYPE=3275,GPOLL=407F,LINE=07A
            TERMINAL    TERM=3275,SELECT=6040,FEATURE=OPRDR,MODEL=3
CLUST079    CLUSTER     CUTYPE=3271,GPOLL=407F,LINE=079
            TERMINAL    TERM=3277,SELECT=6040,FEATURE=OPRDR,MODEL=2
            TERMINAL    TERM=3277,SELECT=60C1,FEATURE=OPRDR,MODEL=2
            TERMINAL    TERM=3277,SELECT=60C2,FEATURE=OPRDR,MODEL=2
            TERMINAL    TERM=3277,SELECT=60C3,FEATURE=OPRDR,MODEL=2
            TERMINAL    TERM=3277,SELECT=60C4,FEATURE=OPRDR,MODEL=2
            TERMINAL    TERM=3277,SELECT=60C5,FEATURE=OPRDR,MODEL=2
            TERMINAL    TERM=3277,SELECT=60C6,FEATURE=OPRDR,MODEL=2
            TERMINAL    TERM=3277,SELECT=60C7,FEATURE=OPRDR,MODEL=2
            RDEVICE     ADDRESS=078,DEVTYPE=3705,ADAPTER=BSCA,          X
                        BASEADD=0B0,CLUSTER=CLUST078
            RDEVICE     ADDRESS=07A,DEVTYPE=3705,ADAPTER=BSCA,          X
                        BASEADD=0B0,CLUSTER=CLUST07A
            RDEVICE     ADDRESS=079,DEVTYPE=3705,ADAPTER=BSCA,          X
                        BASEADD=0B0,CLUSTER=CLUST079
```

In this configuration, if the 3271 cluster control unit is on line 078 there are six display devices and two printers supported. If the 3271 cluster control unit is on line 079, eight display devices and no printers are supported. Display devices can be interchanged among resource addresses allocated to display devices and printers can be interchanged among resource addresses allocated to printers; but a printer cannot be attached at an address defined for a display device and vice versa.

FIG. 3.6 CONFIGURATION EXAMPLE

language following the system conventions, integrate it into the existing system, and modify the configuration process to recognise the new terminal type together with any additional attributes which must be specified.

3.4.2 High Level (Generative) Approach

The BURROUGHS Network Definition Language (NDL) (3.15 and 3.16) is an alternative (and unique) approach to communications network configuration. NDL is a high-level language which not only specifies the physical components comprising the network, but also the logical attributes of each component and the functional behaviour of the network (the way each line is controlled).

The physical description is used to conditionally include code which is dependent on the physical characteristics of the devices (eg. synchronous or asynchronous, switched or permanent lines). Thus, as in the configurative approach, a tailored operating system is generated to meet the requirements of the defined network.

Where NDL differs significantly from the previously described approach is that the logical attributes of each device and the functional operation of each line and station are also defined at configuration time and compiled into executable code performing the specified functions. Conceptually, NDL is not just a systems programming language used in place of assembly language, but rather an application description language which is used to direct a program generator. Although NDL contains some traditional programming constructs, the statements are oriented towards the specific requirements of the data communications environment. The NDL language is essentially a 'host' language which is used to control the

flow between applications-oriented primitives, analogous to many graphics systems which consist of a set of sub-routines embedded within a FORTRAN host program. The 'flavour' of NDL is best illustrated by the example of Fig. 3.7.

3.4.2.1 Description of NDL

This section contains a simplified description of the distinctive features of NDL; the full definition is contained in references (3.15) and (3.16).

An NDL 'program' contains two types of statements. These are:

- The *configurative* definitions, which define the physical network structure and attributes, and
- The *procedural* control and request sections, which define the processing algorithms.

LINE, STATION and MODEM definitions are used to describe the physical configuration and the attributes of the network components.

TERMINAL definitions describe the basic physical attributes of each terminal (speed, transmission mode, etc.) for configuration consistency checking, and a large number of logical or functional attributes.

CONTROL sections are responsible for cyclically selecting each station on the line, performing line scheduling protocol functions, and initiating message transfers. The CONTROL section must perform transmit and receive operations for polling and acknowledgements, but input and output of

```
CONTROL POLL:

        LINE (QUEUED) = TRUE.                   % BUSY

NEXT:   IF STATION > 0
        THEN BEGIN
           PAUSE.                               % GIVE OTHERS TIME
           STATION=STATION-1.                   % NEXT STATION
           IF STATION (VALID)                   % ASSIGNED ?
           THEN IF STATION (READY)              % ACTIVE ?
              THEN IF STATION (QUEUED)          % OUTPUT QUEUED ?
                 THEN INITIATE REQUEST          % YES - TRANSMIT IT
              ELSE IF STATION (ENABLED)         % INPUT ALLOWED ?
                 THEN INITIATE ENABLEINPUT.
           GO TO NEXT.                          % TRY NEXT STATION

        STATION = MAXSTATIONS.                  % RESET STATION PTR
        IF LINE (QUEUED)                        % WORK TO DO ?
        THEN GO TO NEXT.                        % NO - WAIT
        IDLE.

REQUEST POLLINPUT:

        INITIATE TRANSMIT.
        TRANSMIT EOT.                           % POLL
        TRANSMIT ADDRESS (TRANSMIT).            % THE
        TRANSMIT POL ENQ.                       % TERMINAL
        FINISH TRANSMIT.                        % TURN LINE ROUND

GETIT:  INITIATE RECEIVE.                       % GET RESPONSE
        RECEIVE (1 SEC) [ TIMEOUT:DEAD ].       % NO REPLY - DEAD
        IF CHAR = SOH
        THEN BEGIN
           INITIALISE BCC.                      % SOH RESETS BCC

MSG:       RECEIVE ADDRESS (TRANSMIT)
              [ ADDERR:GARBAGE ].
           RECEIVE STX.

TXT:       RECEIVE TEXT [ ETX ].                % DATA ENDS WITH ETX
           RECEIVE BCC [ BCCERR:GARBAGE ].
           TERMINATE LOGICALACK (RETURN).       % DELIVER MESSAGE

ACKIT:     INITIATE TRANSMIT.                   % TURN LINE ROUND
           TRANSMIT ACK.                        % SEND ACK
           FINISH TRANSMIT.

           INITIATE RECEIVE.
           RECEIVE (1 SEC) [ TIMEOUT:DEAD ].
           IF CHAR = STX                        % ANOTHER MSG ?
           THEN GO TO TXT.                      % YES - GET IT
           END.
        IF CHAR = EOT                           % NO MORE
        THEN TERMINATE NOINPUT.                 % MESSAGES

GARBAGE:
```

FIG. 3.7 NDL EXAMPLE

messages may only be performed by REQUESTs, which are invoked by the CONTROL section.

A REQUEST is initiated to transmit or receive a message for a particular station. Text is transferred between the message buffer and the communications line, either 'en bloc', or character by character with program-specified framing. When the request terminates, its CONTROL is reinitiated.

3.4.2.2 Evaluation of NDL

Conceptually, NDL is a powerful and flexible tool for implementing a variety of communications protocols and terminal interfaces. In order to evaluate its effectiveness in these areas, the implementation of the HDLC data link protocol and a virtual terminal interface were investigated. Deficiencies were noted in the following areas:

- Data Structuring Facilities

  The programmer cannot define and use program variables, as in other languages such as PASCAL, but is restricted to a set of predefined byte and bit variables known as TALLYs and TOGGLEs respectively. The NDL programmer must map the program's working storage onto these variables in the same way that an assembly language programmer would use the machine registers.

- Program Structuring Facilities

  NDL does not implement a subroutine or procedure mechanism. This omission may be explained, if not excused, by hardware limitations of the processor.

NDL only implements the 'IF' and 'computed GO TO' control statements. A decision table mechanism or enhanced 'CASE' statement would have significantly increased program clarity and maintainability by avoiding deeply nested decisions, and could easily be compiled into a reasonably optimal decision sequence.

- CONTROL/REQUEST Functionality

Although a RECEIVE REQUEST may be initiated at any time, a TRANSMIT REQUEST can only be implicitly initiated by an output message queued for the station. Therefore non-data frames must be transmitted from the CONTROL level but data frames may only be transmitted from a REQUEST.

- Control of Message Queueing

The NDL system provides no facility for the temporary queueing of message which have been output until they are acknowledged. Any protocol which allows more than one outstanding acknowledgement cannot be implemented using NDL facilities alone.

- Framing Functions

NDL provides the TRANSMIT TEXT and RECEIVE TEXT statements which process an entire message. However, if any character processing must be performed for transparency purposes, the message must be explicitly processed character by character. This 'programming language' approach restricts the implementation of the framing functions, since it specifies not only what is to be done, but also how it is to be performed.

- Terminal Virtualisation

  NDL provides within the terminal definition the ability to define a number of logical and functional attributes of a terminal, such as the page size and control characters for common functions (Backspace, Line Delete, End of Line, Home, Clear, etc.). However, NDL places no semantic interpretation on these definitions; even the simplest formatting functions must be explicitly programmed within the request, along with the line protocol framing functions. Many of the attributes which may be defined are 'for documentation only', and of no use to the programmer.

- Full Duplex Operations

  NDL implements the FORK, WAIT and CONTINUE constructs for the interaction of two processes controlling a full duplex line. The reference manual includes explicit warning of potential deadlock problems due to implicit timing dependencies between these operations.

3.4.3 Summary of Implementation Approaches

While the configurative approach is completely adequate for specifying a network within the confined environment of a particular manufacturer's product line, it is severely limited in the more general case where it is necessary to support an arbitrary range of devices and protocols. The generative approach of NDL conceptually offers significant advantages, but these are not realised in the implementation which in practice, apart from 'syntactic sugar', offers little more functionality than a well designed library of subroutines or macros in an assembly language environment.

3.5 DIRECTIONS FOR DEVELOPMENT

Provided that the communications system functions reliably and at low cost, the internal mechanisms are really of no interest to the user. Rather, the user's view of the network will be determined by the characteristics of the interface between the user and the devices connected to the network. In the same way, an applications programmer views a computer in terms of the job control language and compilers rather than the underlying hardware architecture.

The development of local communications network technology has concentrated on the development of better data communications technology, whereas computer manufacturers have emphasised the support of existing equipment and provision of user-oriented facilities.

If local computer network technology is to progress from the experimental stage to being a general tool for systems structuring, user-oriented facilities must be provided so that the local network can be configured to adapt to the applications environment, rather than the environment adapt to the interface conventions of the network. The most promising approach appears to be through the use of a high level language specification of the terminal attributes and communications protocol functions which is used to generate a customised communications processor operating system for the particular environment.

4. THE NETWORK INTERFACE MACHINE: CONCEPTS AND DESIGN

The previous chapter examined the technology available to implement local computer networks. This is well developed, with the major impediment to a more widespread usage of local networks being the difficulties encountered in connecting devices to the network in such a way that they can readily interact with each other.

A key development in the design of communications networks is the concept of an *architecture*. The communications architecture defines the logical structures and relationships of all functional components within the system and establishes a set of rules and guidelines which determine how the logical structures are applied to the physical system through specific interfaces and protocols.

This chapter first examines the concept of a *Network Interface Machine* (NIM), designed to facilitate the interconnection of a variety of terminals, computer systems, and external communications links to a local network. From these conceptual requirements the functional divisions and the logical structure of the system are developed. Finally implementation approaches, both of hardware and software, are discussed.

4.1 FUNCTIONAL REQUIREMENTS

The function of a local network is to interconnect local mainframes and a proliferation of minicomputers, microcomputers, terminals and peripheral devices, and communications links or networks.

As noted in the previous chapter, existing experimental networks vary greatly in transmission methods, and in mechanisms for connecting user devices to the network. As communications technology continues to evolve, new transmission methods and protocols will be utilised, but the functionality of the local network will, to a great extent, be determined by the design of the network interface. The network interface capabilities and the methods of their implementation primarily determine the 'personality' of the network - they affect the ease of use, transparency, vendor independence, cost and flexibility of the network.

The network should aim to provide a 'friendly' interface, so that the attached devices may be unified into a complete system with minimised impact on each device. If the network interface is isolated as much as possible from the user devices and the underlying data transport services, the network will have the required versatility and capabilities to provide transparent interconnections for a wide range of devices and transport network architectures.

Specific objectives for a local network interface are:

- To provide an interconnection between a large number of terminals and computers of different types. With falling prices and increasing acceptance of interactive computing the number of terminals may approach the hundreds, while the number of computers ranging from microprocessors to mainframes may reach the tens.

- The ability to support general terminal to computer communications including mapping between the characteristics of the various terminals and computers (terminal virtualisation).

- The ability to interface terminals to the network inexpensively. With terminal prices in the order of a thousand dollars, this should not exceed a few hundred dollars for an individual terminal, and reduce when several terminals are clustered to share an interface.

- The ability to support computer to computer communication of binary data at high transfer rates.

- The ability to interface computers by a multiplexed physical connection supporting many virtual connections.

- The ability to off-load network-specific interface software from connected computers into the network interface.

- The ability to access remote resources via gateways to other communications networks.

The Network Interface Machine implements a 'black box' approach to the network interface through a set of *adaptor* modules, each performing specific interface functions. These adaptor modules provide device-independent interfaces to the network controller, which manages the logical connections or sessions between paired end-users. In a full network the local node controller uses the transport network to communicate with the destination node controller. However, a NIM may also be configured in such a way that transport network facilities are not required, but the machine simply implements a transformation function utilising two or more adaptor modules. Such a configuration (Fig. 4.1-B) may perform the functions of:

- A *Front-End Processor* (FEP) for an existing computer,

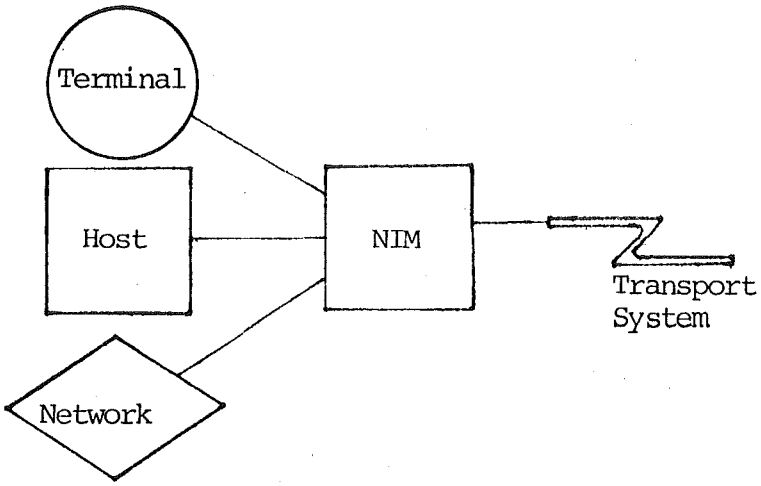


FIG. 4.1-A CONCEPTUAL NIM

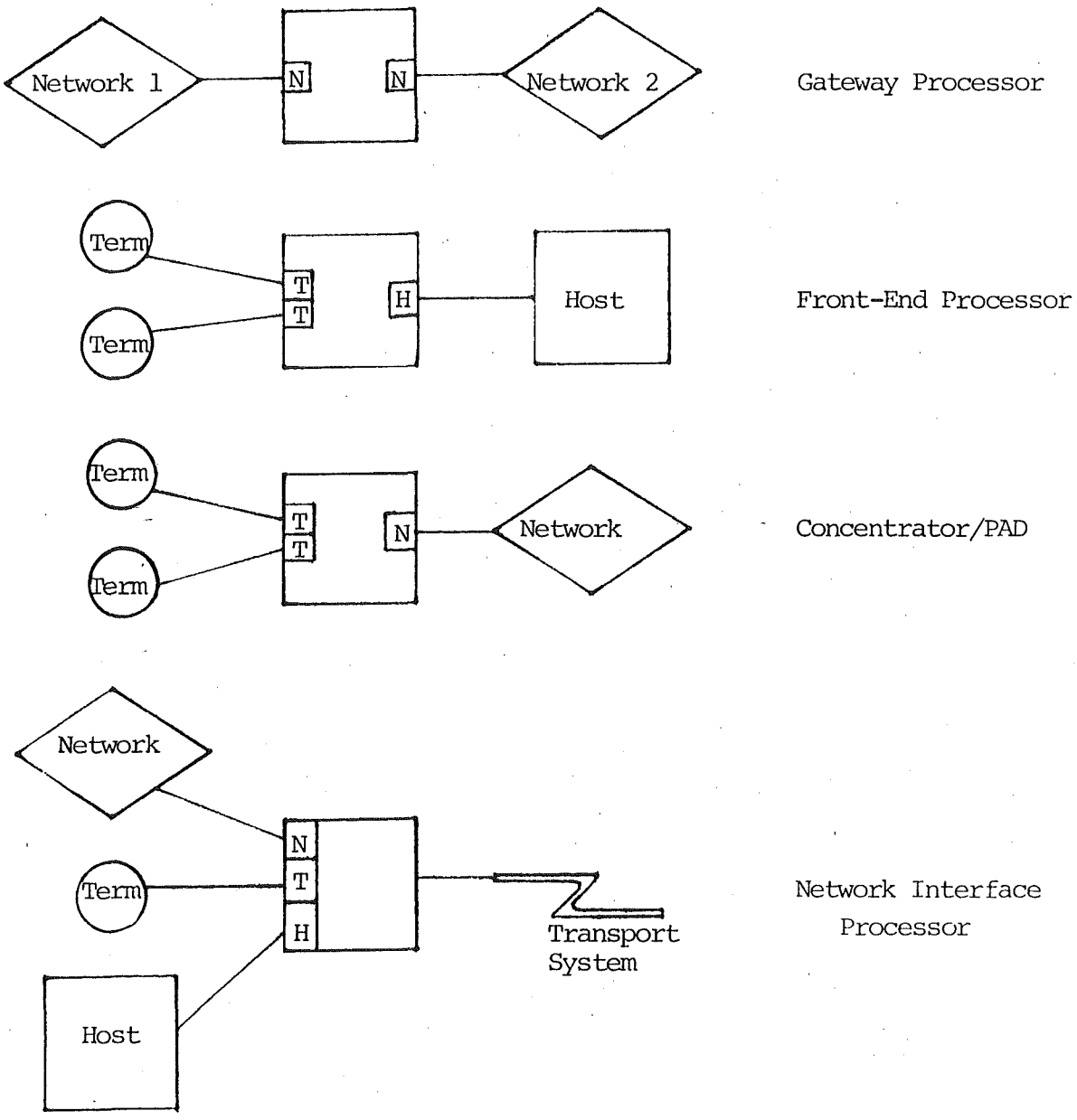


FIG. 4.1-B POSSIBLE CONFIGURATIONS

- A *remote concentrator* or a *Packet Assembler/Disassembler* (PAD) interfacing terminals to a public data network,

  or

- A *gateway* transformation between two communications protocols.

In the following subsections the functional requirements of each component is examined in greater detail.

4.1.1 Transport Services

The network access levels require a well-defined interface with a communications sub-system or transport facility. The term *transport* emphasises that the fundamental function of this facility is to move data from one place to another. Ideally it should be error-free and incur no transfer delays.

The definition of the transport interface should make as few assumptions as possible about the specific techniques used to carry data within the transport facility. Even though some techniques are more appropriate than others with current technology, nothing should prevent the adoption of different techniques, or new technologies (such as fibre optics) when they become available. The internal mechanism should not affect the transport interface.

Basic functions of the transport end-to-end control level are:

- To implement a standard interface between the providers of the transport services and the users of the transport services.

- To perform the mapping between user messages and internal packets by segmenting and reassembling messages where necessary.

- To provide end-to-end control and error checking ensuring the sequential and reliable transfer of data.

- To transform external addresses to internal network addresses.

When the functions provided by the transport services are not exactly those of the interface specification, an additional layer can be added *wrapping* the lower level services. The wrapping layer performs the additional or modified functions to implement the transformation between the interface specification and the actual transport services.

Transport facilities may be grouped into two classes, known as Virtual Circuits (VC's) and Datagrams (DG's).

- A *virtual circuit* is a logical path which is established between end-entities for the purposes of exchanging data. Once the VC is established, data may be transferred between the entities by refering to the path name rather than the full end-point addresses (known as *abbreviated addressing)*. Virtual circuits deliver information in the same sequence in which it was sent, and perform error detection and recovery functions to ensure reliable transfer.

- A *datagram* is a self-contained packet of information which is carried to the specified destination without reference to any other packet, or prior establishment of a data path. Each datagram transfer is a self-contained transaction, and as such contains the full source and destination addresses. Datagrams provide a very simple transport facility, without mechanisms normally associated with the orderly and reliable transfer of information.

These two types of services may be made equivalent to each other by the addition of a wrapping layer.

A datagram service may be transformed into a virtual circuit service through the addition of a wrapping layer which performs the additional functions of call establishment and clearing, packet ordering during data transfer, and error and flow control. During the information transfer phase the abbreviated address is mapped onto the full addresses which were stored at 'call establishment' time.

A virtual circuit service may similarly be transformed into a datagram service. The virtual circuit service may be wrapped by an additional layer which initially establishes virtual circuits between nodes. Datagrams may then be passed transparently over these virtual circuits which are simply used in place of physical circuits.

4.1.2 Session Control

To create an environment in which end users can freely communicate, supervisory services are required to support the logical connections between network users. The provider of these services is commonly known as the Network Control Program (NCP). The NCP functions may be centralised or, more

desirably, distributed among the network nodes. With distributed network control, each NCP manages the resources within a node and communicates with NCP's in other nodes forming a distributed operating system controlling the logical connections and interchange of data between paired end-users. This section discusses the fundamental tasks which must be performed by a NCP, and a general implementation approach.

The basic functions which must be performed by a NCP are:

- The establishment, maintenance and termination of connections,
- Information transfer and flow control across connections,

  and

- error handling.

The fundamental function is that of establishing, maintaining and clearing the logical connections, known as sessions, between users. There are two aspects to session establishment; the addressing of termination points and the actual procedures used.

An *addressing mechanism* is required to allow users to designate each other within a global address space. The addressing scheme should be rich enough to allow not only intra-network addressing, but also the ability to access host systems and interconnected networks. A simple linear address space requires each node to be aware of the port configuration of every other node. This

- Is inflexible as a change in an individual node's configuration affects the whole network,

- Is expensive as the space required to store the configuration grows non-linearly with increasing network size,

- Imposes architectural restrictions on the ability to access external networks having a dynamic address space.

A *hierarchical address space* views the global network as a collection of subnetworks. The address space consists of two portions; the first identifying the subnetwork, and the second identifying the particular port within the subnetwork. This is flexible since:

- An individual node is required to interpret only the first-level address of each subnetwork, and the second-level addresses for its own ports only,

- It allows a dynamic address space, and different forms of intra-node addressing. Because the second-level address is meaningful only within its subnetwork, a variety of symbolic port addressing conventions may be used, facilitating inter-network communication.

- It provides an intra-network addressing mechanism which is applicable to both bus (broadcast) and mesh (switching) architectures since routing functions need only interpret the first portion of the address to recognise packets addressed to this node, or to forward a packet towards the destination.

A connection protocol is required to specify the procedures by which NCP's interact to establish sessions for users. In principle only two commands are required; a connection command to establish a session and a disconnection command/response to

terminate a session or refuse a connection request. The positive response to a request to establish a session may be a reply of the same command. A disconnect command may be used as a negative response to a connection request, or a positive response to a disconnection request. The connection protocol must also cater for error situations ranging from contention for a termination point to unrecoverable link errors or failure of the remote node.

The second function of the NCP is controlling the flow of data across the sessions. This flow control is required in two directions; horizontally to control the interaction between the end-users, and vertically between each end-point and the transport network to prevent network congestion.

The simplest method of flow control is to ensure that the next message may not be sent until an acknowledgement has been received in response to the previous message. This obviously limits the maximum transfer rate of each individual session, and reduces overall efficiency because every data transfer incurs the overhead of its corresponding acknowledgement. This scheme was used by the original ARPANET protocols, with a source node requiring a 'Ready For Next Message' (RFNM) response from the destination before transmitting the next message, but found to be too restrictive.

A slightly more complex method which overcomes these limitations is to establish a *credit limit* specifying the maximum number of messages outstanding over a connection at any one time. When the credit limit has been reached, a 'go-ahead' response is required from the receiving end before any further messages may be sent. It may also be necessary to provide a means to acknowledge messages without raising the credit limit, and to synchronise the end-points by waiting until outstanding messages have been completely processed.

In addition to normal data transfer, a mechanism is required for the expedited transfer of control and status information which is not restricted by the normal flow control mechanism. An example is that of a time-sharing system user who wishes to abort a program stuck in an infinite loop. The abort request must have precedence over any data queued, and be passed directly to the host's operating system rather than being queued. This may be achieved either through the use of a high priority data transport service (such as the Interrupt in X.25), or by the NCP's communicating over a special connection.

To ensure that session control does not assume the existence of transport facilities which may not be provided, such as virtual circuits, the multiplexing of sessions between pairs of nodes should be implemented by session control independently of the transport network services. This allows the freedom to allocate a separate virtual circuit for each session, or to multiplex sessions over a single circuit when necessary.

The final function of session control is the handling of errors within different components of the system. This has two aspects, detection and recovery.

Error situations may be detected directly from status conditions, or indirectly by failure to observe a mutually-agreed procedure. An example of the former is loss of carrier on a communications line, while of the latter is failure to respond within a predetermined time period. When an error is detected, there must be a reliable way of returning to normal operation. Two approaches are simply to terminate any activity affected by the failure, or to attempt recovery when the failing component returns to normal operation. Hopefully error situations are sufficiently rare that the termination of affected activities is acceptable.

4.1.3 Terminal Interface Architecture

The preceding chapters have shown that there is a vast difference between the provision of a physical connection and meaningful interaction or dialogue.

The diversity of terminal characteristics results in compatibility problems; some device-specific adaptation must be implemented whenever a new type of terminal is to access a computer system.

With the conventional star-like terminal network, which provides access to a centralised host, this is feasible since the terminals are likely to be from the vendor's product line and supported by vendor-supplied software.

Within the distributed network environment, however, the problem will become much more evident as a wider selection of terminal devices will each access a variety of hosts. It is no longer feasible to perform an explicit adaptation for each terminal to each host, as the task of interfacing *M* terminals to *N* hosts assumes the proportions of *M*N* (see Fig. 4.2-A).

A much more manageable solution is to specify common network-wide procedures for the interaction between terminals and applications. These procedures are commonly known as a *Virtual Terminal Protocol* (VTP), and the physical device together with its adaptor may be viewed as an abstract or *virtual terminal* (Fig. 4.2-B). The mapping of a particular device onto the virtual terminal need only be made once, but all applications which communicate with the device must adhere to the virtual terminal protocol.

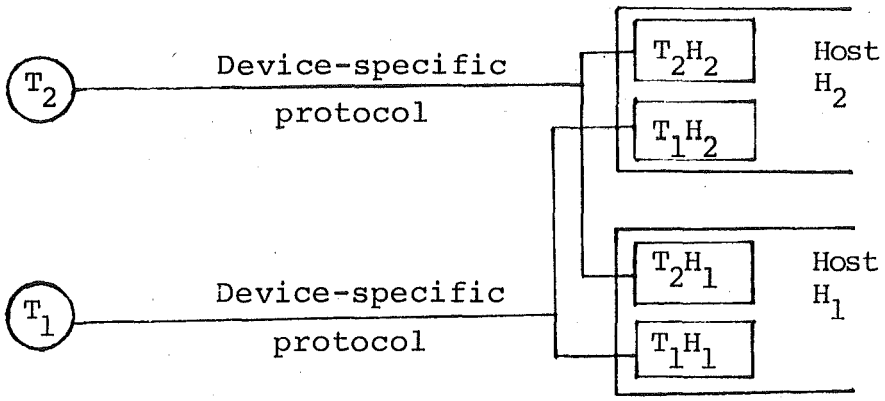


FIG. 4.2-A M x N INTERFACE PROBLEM

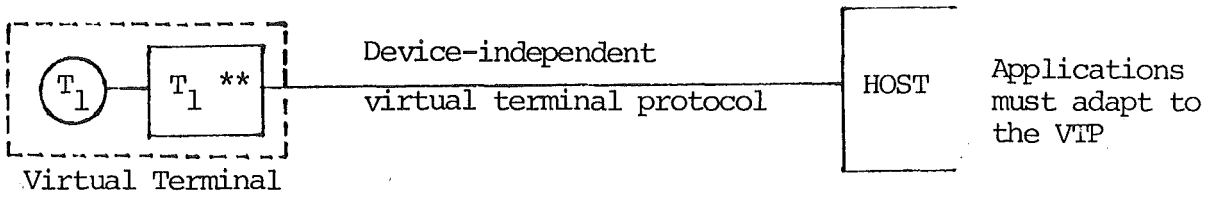
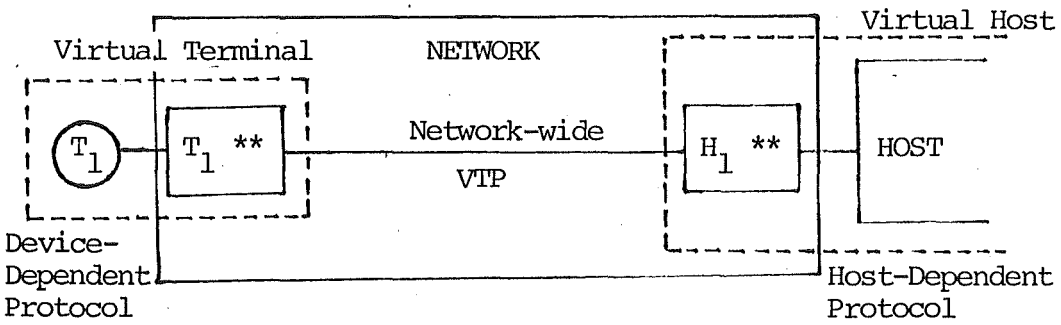


FIG. 4.2-B TERMINAL VIRTUALISATION



** Virtualiser

FIG. 4.2-C DEVICE AND APPLICATION VIRTUALISATION

The most flexible solution is to implement two adaptions - one at each end of the connection - and perform a mapping not only between the physical device and the VTP, but also between the VTP and the application (Fig. 4.2-C). In the same way that the virtual terminal consists of the device and its adaptor, the *virtual application* consists of the actual application and its adaptor to the VTP; if the application implements the VTP directly then the adaptor is null. Conceptually this approach results in an overall compatibility between all applications and terminals.

The terminal adaptor is comprised of two levels:

- The higher *virtualisation* level performs the mapping between device-dependent and device independent formats in a similar manner for all terminal types.

- The lower *communication/emulation* level varies in functionality according to the type of terminal. Synchronous terminals require the implementation of a data-link control protocol, while asynchronous terminals require message delimiting and editing functions.

4.1.4 Host Interface Architecture

The two factors to be considered in the host interface are:

- The implementation of the physical communication path between the NIM and the host processor, and the integration of this within the host system software structure.

- The implementation of the adaptor from host-specific protocols to the network-wide device independent protocol.

A major consideration must be to minimise impact on the host. On many computers the implementation of a new device handler or protocol handler is a major undertaking. In some cases this may be justified, but in general the interface should make use of existing facilities wherever possible.

The two basic approaches to the physical interconnection may be classified as memory or communications oriented.

The memory interface functions in the same manner as a block transfer I/O device. It requires a hardware interface to the host processor's channel or Direct Memory Access (DMA) controller. This approach is very efficient as it avoids host processor intervention during data transfer, but it almost certainly involves some hardware development and (possibly considerable) software enhancements to the host's operating system.

The communications interface functions through the host's normal communications subsystem. If the host provides an X.25 interface (which many vendors have developed or are developing) then it is a natural choice, particularly as several logical connections may be multiplexed over a single physical link. An alternative is the configuration of a BISYNC (or similar protocol) multi-point line on the host, with each station representing a logical connection. A limited access may even be provided by a single asynchronous line.

While it would be most desirable for all programs to interface to the VTP directly, an interface which is compatible with existing programs and protocols must be provided to allow a gradual migration to the new services. The host adaptor should provide a translation between each

hosts' normal terminal protocol and the network-wide VTP, performing such functions as character-code conversions and control sequence mapping. Host impact may be minimised by performing these conversions within the network interface processor rather than in the host itself.

4.1.5 Network Interface Architecture

The ISO Open Systems Interconnection Architecture (see 3.2.2.1) proposes a layered model defining the scope and relationships of functions from the physical level through to the applications level. In general, while networks are not compatible with each other, and therefore cannot be directly interconnected, the basic functions are usually similar. Differences between individual networks may be reconciled with overall compatibility (common conventions between systems) by viewing commonality in terms of these functions, leaving freedom for different internal implementations.

Networks may be interconnected at the frame, packet or higher levels, the actual interface between the networks being known as a *gateway*. The minimum function of a gateway is to forward data and control functions from one network to another. Since these functions are not directly equivalent, it is necessary to either select a common subset of the services provided by both networks, or to add additional functions to one (or both) networks to bring them to an equivalent level.

Three levels of gateway participation may be distinguished. For each protocol level the gateway may:

- *Terminate* elements of the protocol

  When one network provides services which have no equivalent in the other, the gateway must act as an

end-point for these services. Protocol levels below the interface must also be terminated. For example, a gateway at the X.25 Packet level must terminate the HDLC level functions.

- *Translate* elements of one protocol to the other

  When both networks provide equivalent services, which differ only in their implementation, the gateway must perform a translation between these services.

- *Be Transparent* to elements of the protocol

  When both networks provide identical services, these may be passed directly across the gateway without intervention.

4.2 SOFTWARE ARCHITECTURE

The preceding sections have examined the functional requirements of the Network Interface Machine. This section gives an overview of the software architecture developed to satisfy these requirements, and the flow of messages through the system. The detailed design of each component is considered in the following chapters.

4.2.1 Hierarchical Layers

The Network Interface Machine is comprised of four functional entities. These are:

- The *Transport System*, which provides inter-node communications facilities

- The *Session Control*, which controls the connections and the structured interchange of data between network ports

- The *Virtualiser*, which performs a transformation between the network-wide device independent protocol and the particular device protocol.

- The *Access Interface*, which implements the logical and physical interface between the network interface machine and the external devices.

The virtualiser and the external access interface are collectively referred to as a *termination system*, since they provide the logical termination points of sessions.

4.2.1.1 Transport Network

Since the primary emphasis of the NIM is the provision of higher level facilities above those of the basic data transport network, detailed discussion of the structure of the transport network is confined to its interface with session control.

The transport network may be viewed as consisting of four sublayers, which are:

- Framing

  Receiving and transmitting data over the physical data link.

- Data Link Control

  The procedural functions of flow control, error detection and recovery, etc.

- Routing Control

  The addressing and routing functions of intra-network switching points.

- End-to-End Control

  Provides message transport services between source and destination nodes, and the interface to the users of the transport services.

4.2.1.2 Session Control

Session control performs the centralised switching function within a node, controlling the logical connections or liasions between the logical ports representing network user-processes. The three levels of interaction which occur are:

- *Above* to the port processes

  Session control is responsible for the transfer of messages between paired logical ports, which may be in the same, or different nodes. Port messages may require segmentation into fixed-size packets for transmission over the network, and reassembly at the destination.

- *Below* to the transport network

  When the destination port is in another node, the transport services must be used to deliver the message to the destination.

- *As a Peer* to session control in other nodes

  Session control within a node must interact with the

session control of the other nodes to establish sessions and control data flow between the session ports.

4.2.1.3 Virtualisation

Virtualisation performs the translation between the network-wide device-independent protocol and the many device-specific protocols. Although primarily associated with terminal devices, some degree of virtualisation is required whenever two dissimilar devices are interfaced, whether they be computers, communications networks, or peripherals.

4.2.1.4 Access Interfaces

The access interface implements the logical and physical communications path between the Network Interface Machine and the external device. There are two categories of interface; the character oriented terminal interface and the message oriented communications interface. The latter encompasses all interfaces requiring a message protocol, including host channel interfaces, synchronous terminal line protocols, and communications network interfaces.

4.3 HARDWARE ARCHITECTURE

This section examines alternative hardware approaches to the implementation of the Network Interface Machine. Two alternatives are considered, minicomputers and multiple microprocessors.

4.3.1 Minicomputer Hardware

Using a conventional minicomputer-based communications processor, the majority of the work is performed by the central processor, with functionally distributed hardware handling only the assembly and disassembly of characters for asynchronous lines, or possibly frame recognition for synchronous line protocols.

This approach has the following characteristics:

- An operating system of some complexity is required to share the processor and resolve contention between extremely time-critical (but short duration) events such as interrupt processing, and the more time-consuming message manipulation processes.

- Expansion is difficult. Increased capacity requires three additional resources:

  - Line Hardware

    Additional line hardware may be added relatively easily. Most processors have an adequate number of bus interface slots. However, when these slots have been used, a bus expander is required, which may be expensive.

  - Memory

    Memory may be added up to the address limits of the machine. To go beyond this generally requires both the installation of a new processor with address-mapping hardware, and operating system changes to maintain a logical address space for each process.

- Processor Capacity

  Processor capacity may be extended to some extent by the addition of cache memory or faster main memory, or an enhanced instruction set, but beyond this an exchange of processors is required.

- A hardware or software fault in the central processor may make the entire system inoperable.

The basic problem is that the nature of the workload (a small amount of work replicated many times) is not reflected by the modularity of the hardware, and neither is the varying nature of the workload (time-critical interrupt processing combined with normal processes).

4.3.2 Multi-Microprocessor Hardware

The alternative approach is to distribute the processing functions between a number of processors, so that the hardware modularity matches the modularity of the workload. The combination of both modular hardware and software can provide a 'building-block' approach to communications processor design, suitable for the construction of both special-purpose and general-purpose communications systems which can be restructured with relative ease.

The advantages of this approach are:

- Functional Isolation

  Since software functions can be allocated to particular hardware modules, there is an increase in system security through hardware-enforced

containment and protection. This results in increased maintainability and reliability of the system because the interactions between components are minimised.

- Expandibility

  Since hardware can be added in unit increments, 'quantum jumps' in hardware resources can be avoided.

- Throughput

  Multi-processing can result in an increase in throughput. This can be realised through:

  - *Parallelism*

    Similar functions can execute in parallel on simple replicated hardware.

  - *Pipelining*

    When a function consists of a series of chained stages, the stages can be split across processors allowing more than one function chain to be processed concurrently.

  - *Reduced Contention*

    If functions are dedicated to particular hardware modules, the executive overhead required to share global resources may be largely avoided.

- Modularity

  A standard interface between the providers and users of services allows the actual implementation of these services to be varied without affecting higher levels.

- Functional Design

  The allocation of specific functions to hardware modules may allow the development of hardware optimised for that function, rather than using general-purpose devices less efficiently. An example is the variety of SDLC chips available.

4.4 IMPLEMENTATION APPROACHES

This section examines possible hardware and software implementation approaches for the Network Interface Machine, resulting in the selection of a 'system generator' compiler for multi microprocessor hardware.

4.4.1 Design Objectives

The design objectives of the Network Interface Machine software are:

- The global nodal configuration should be separate from the local configuration. A change within a node should only affect that node, and a change to the nodal configuration should only require the generation and reloading of updated routing tables for nodes with no local configuration changes, rather than a complete recompilation of the code for every node.

- The software structure should allow functions to be implemented on modular hardware. It should be possible to implement a simple NIM on minimal hardware, yet to distribute functions over hardware modules as the nodal complexity expands by specifying simple configuration changes, rather than detailed program amendments.

- The network interface should support a broad range of devices simply by specifying their characteristics, rather than detailed data formatting algorithms for each device.

- The network interface should be adaptable to a wide variety of communications protocols, which are implemented as far as possible by specifying the functions which are to be performed, rather than the step by step actions required to implement each function.

- The software must make efficient use of the hardware resources.

4.4.2 Hardware

Until recently the most expensive component of a computer system has been the central processor, with a collection of smaller processors costing more than a single larger processor of equivalent performance. The development of Large Scale Integration (LSI) semiconductor technology has completely reversed this balance. It is now possible to produce microprocessors at minimal cost, and to interconnect them using standard system-level components.

The Intel MCS family of products has been chosen to implement the Network Interface Machine. Although individual items from other manufacturers may offer technical advantages, the capabilities of the Intel system-level components exceed those of other manufacturers, and compatible boards are available from other sources. In addition considerable experience has been accumulated in the use of these products. The MCS family includes:

- A multiprocessor bus providing address mapping, locking over the duration of global memory accesses, and capable of supporting up to 16 processors.

- 8 and 16-bit microprocessor boards which may be intermixed. To avoid performance degradation due to bus-access conflicts, each microprocessor has local on-board memory, which may be dual-ported to also allow global access.

- A family of programmable 'Universal Peripheral Interfaces' (UPI's) which may be programmed to perform a variety of I/O functions in a standard manner.

- Specialised communications devices, such as USARTS, SDLC controller chips, etc.

4.4.3 Possible Implementation Approaches

Techniques of communications software implementation were reviewed in Section 3.4. Three possible implementation approaches identified for the NIM software are:

- Conventional specialised code

- Table-driven parameterised programs

- 'Systems generator' compilers

4.4.3.1 Conventional Specialised Code

Machine language has been the traditional method of implementing communications software. Careful hand coding can result in near-optimal code. However, program development and maintainence is very expensive and time-consuming, requiring a high level of technical expertise in both communications techniques and the particular computer.

A macro language can provide significant advantages over standard machine language, in that it is possible to implement 'instructions' performing particular communications tasks. Each macro can be designed to generate different code depending on the parameters, resulting in little loss of efficiency, but greatly reduced programming effort. However, a high level of programming expertise is still required.

High level languages have, over the past few years, found increasing acceptance in areas that were once reserved for machine language. For larger machines, compilers are available which can produce code of comparable quality to the best hand-written code, while reducing software development time and increasing maintainability. However, the instruction sets of microprocessors are quite restricted and irregular, which makes compiler optimisation difficult. Indexed addressing limitations become particularly evident when re-entrancy is required, since a sequence of instructions is often required to perform the equivalent of one operation on larger machines.

PL/M is the standard systems-programming language for the 8080 microprocessor. From investigation it was found that hand produced code was generally a factor of 2 better in size and speed than compiled code, and up to 5 times better when extensive table manipulations were being performed.

The alternatives considered above offer increasing ease of implementation, but do not address the primary objective of easy configurability. Of these, macro languages offer the most potential, but they are generally limited to specific computers.

4.4.3.2 Table-Driven Parameterised Programs

Conventional software requires tables to specify the network configuration and limited attributes of the network entities. In general, these tables are used to define occurences of entities (lines, terminals, etc.) which are 'known' by the software, rather than to describe the attributes of new types of entity.

The attributes of different types of terminals could be specified by tables which are interpreted by the virtualisation functions. Tables are very suitable for this situation, since different terminals generally provide common sets of functions differing only in the specific implementation (ie. the control sequences which invoke the function). However, tables are not adequate to parameterise a program for different communications protocols, since the algorithms themselves are variable, rather than being fixed but operating with variable data.

4.4.3.3 'System Generator' Compiler

A *program generator* is designed for a specific application, interpreting user specifications to generate a customised program which performs the specified functions. It may be regarded as a compiler for a very high level language, translating from requirements (*what* is to be done) to the corresponding actions (*how* it is to be done).

Program generation combines the advantages of the previous approaches. The specifications are much more concise and simpler than the generated program, reducing the effort to implement a specific function and increasing the flexibility. Since code is produced for pre-determined functions in a known environment, the efficiency of assembly or macro languages can be obtained, while retaining the machine independence and ease-of-use features of high level languages. Table-driven techniques can be used where appropriate, with the tables being constructed from user-supplied configuration and attribute specification statements. Specialised code can be used for those areas requiring a high degree of flexibility, but at the level of the functions to be performed, rather than their step-by-step implementation. The program generator itself is as portable as the language in which it is written, although the generated code may be dependent on specific hardware.

The initial effort to implement the program generator is higher than that to implement a specific system, since it is not only necessary to code all the run-time routines performing communications oriented functions, but also the specification compiler which translates between the user requirements and the corresponding run-time functions. However, once implemented, the program generator provides a

'tool' by which a variety of specific communications systems can be generated with relatively little effort.

4.4.4 Chosen Implementation Approach

In view of the above, it was decided to design and implement a 'communications system generator' compiler, which would process a high level network specification and generate corresponding communications operating system code for the Intel 8080 and 8741 microprocessors.

The high level language is used to specify:

- The Global Node Configuration

  The global node configuration specifies the nodes, and the communications paths between nodes, which comprise the network.

- The Local Node Configurations

  The local node configurations specify the physical configuration of each node in terms of the access interfaces (communications lines or computer channels), the stations (logical end-users) for each access interface, and the allocation of functions to micro-processors within the node.

- Device Characteristics

  For each type of device that is interfaced to a nodal processor it is necessary to tabulate the device's physical and logical characteristics. These are used to direct the virtualiser's translation between device dependent and network-wide protocols.

- Communications Protocols

  For each type of communications interface the message formats and the protocol procedures must be specified.

Chapters 5, 6 and 7 consider the design of the run-time system and specification language for session control, virtualisation, and the access interfaces respectively, while chapter 8 considers the compiler structure in more detail.

5. SESSION CONTROL

Session Control is the supervisory level of the network interface machine. It manages the establishment of, and the transfer of information over, the logical connections (known as *sessions*) between paired end-users.

This chapter describes the implementation of the requirements specified in Section 4.1.3. Firstly, a common data encoding mechanism used by the higher level network functions is defined. The data structures and processing functions of session control are then described, together with the session management protocols. Finally, the network configuration language is described.

5.1 ITEM DESCRIPTOR BASED PROTOCOLS

Conventional protocols may be categorised into two types, depending on whether character values or sequence position are significant, as previously described in Section 3.1.

At the data transport level, positionally significant (tabular format) protocols are used in the more recently developed protocols virtually without exception. Fixed format headers provide a compact data representation for a predetermined set of functions, particularly when variant definitions of fields are used.

At the network management and higher levels, however, fixed format tables are less suitable. While lower level functions are well defined, with most messages using the entire set, higher level functions may be very open-ended

and individual messages may only use a very limited subset of the available functions. The Item Descriptor (ID) protocol format has been developed for these situations.

5.1.1 Item Descriptor Formats

An ID protocol is based on a serial encoding of the data using highly-formatted control bytes to specify both the *actions* to be performed and the *format* of the data. Each control byte is known as an Item Descriptor (ID).

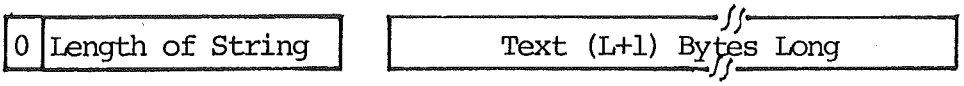
ID's may be used to encode commands or data objects. The three ID formats (Fig. 5.1), distinguished by the setting of the top one or two bits are:

- S *(String)* Format

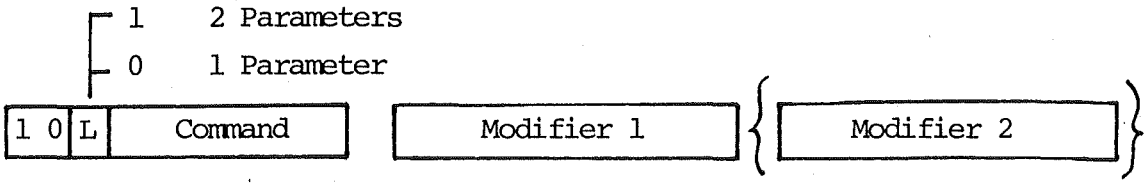
  The S-format ID, used to encode variable length data strings, is distinguished by '0' in the top bit. The remaining seven bits specify the length of the following data string.

- M *(Modified)* Format

  The M-format ID, used to encode commands which are modified by parameter values, is distinguished by '10' in the top two bits. The third bit determines whether one or two parameter bytes follow, and the lower five bits specify the command type. This command type is meaningful only to the specific protocol using the ID formats. The following one or two bytes may be used to specify a variant of the command, or parameters for the command.



S-FORMAT (STRING)



M-FORMAT (MODIFIED)

U-FORMAT (UNMODIFIED)

M-FORMAT PAGE SHIFT

FIG. 5.1 ITEM DESCRIPTOR FORMATS

- U *(Unmodified)* Format

  The U-format ID, used to encode simple unqualified commands, is distinguished by '11' in the top two bits. The remaining six bits specifying the command itself. The U-format ID may often by followed by an S-format string which specifies the object to which the action is to be applied.

The basic encoding scheme, known as a *page*, provides:

- 32 modified commands with 1 or 2 parameters
- 64 unmodified commands
- Arbitrary string data

To provide a virtually unlimited number of commands, a 'page switching' mechanism is used. There are 256 pages, with the page number acting as an implicit extension to the command code. Each page may have a different interpretation of the ID codes. Page switching is accomplished by reserving code 00000 of the M-format ID to select the active page.

5.1.2 Advantages and Disadvantages

The advantages of the ID protocol over fixed format table protocol structures are:

- Global subroutines can be used to encode and decode the ID's since they are in a common format.
- Programming is simplified since look-up tables can be used to encode and decode the ID's.

- Variable length parameters and values are the rule rather than the exception.

- ID codes are extensible. Extra code values may be added with no impact to existing functions.

- Software components can operate independently of fixed parameter formats, since the meaning is distinguished by the codes within the ID's rather than by location within a table.

Possible disadvantages are:

- The data format descriptor takes an additional few bits for each field, which are not needed for table-format protocols.

- The size of buffer required is not known until all the ID's have been built, whereas tables have fixed sizes. Therefore an estimation scheme is required, or a method of dynamic buffer chaining.

- Tables can be accessed randomly, but ID's must be processed serially. If data is to be repeatedly accessed, it may be necessary to build a table or save the buffer contents for later access.

For higher-level applications-oriented protocols, the advantages far outweigh the disadvantages.

5.2 SESSION CONTROL STRUCTURE

Session control is centered around:

- *Tables* which describe the static and dynamic configuration of the network.

- *Protocols* which implement the session control services.

5.2.1 Data Structures

The structure and the relationships of the session control tables are illustrated in Fig. 5.2. The Global and Local Configuration Directories (GCD and LCD) are built at network generation time and are used for session establishment, while the Session Table maintains the dynamic environment for each active session.

5.2.1.1 Global Configuration Directory

The GCD contains the symbolic name of every Termination System within the network, and its corresponding nodal address. An identical copy exists within session control of each node. The GCD is used to construct the first part of a network address for inter-node communication.

5.2.1.2 Local Configuration Directory

Each LCD describes the logical configuration of the node in terms of the Termination Systems (TS's) and ports within the node.

For each Termination System the LCD contains:

- The symbolic name of the TS

- The internal TS address used by session control. This corresponds to the GCD entry index for the TS

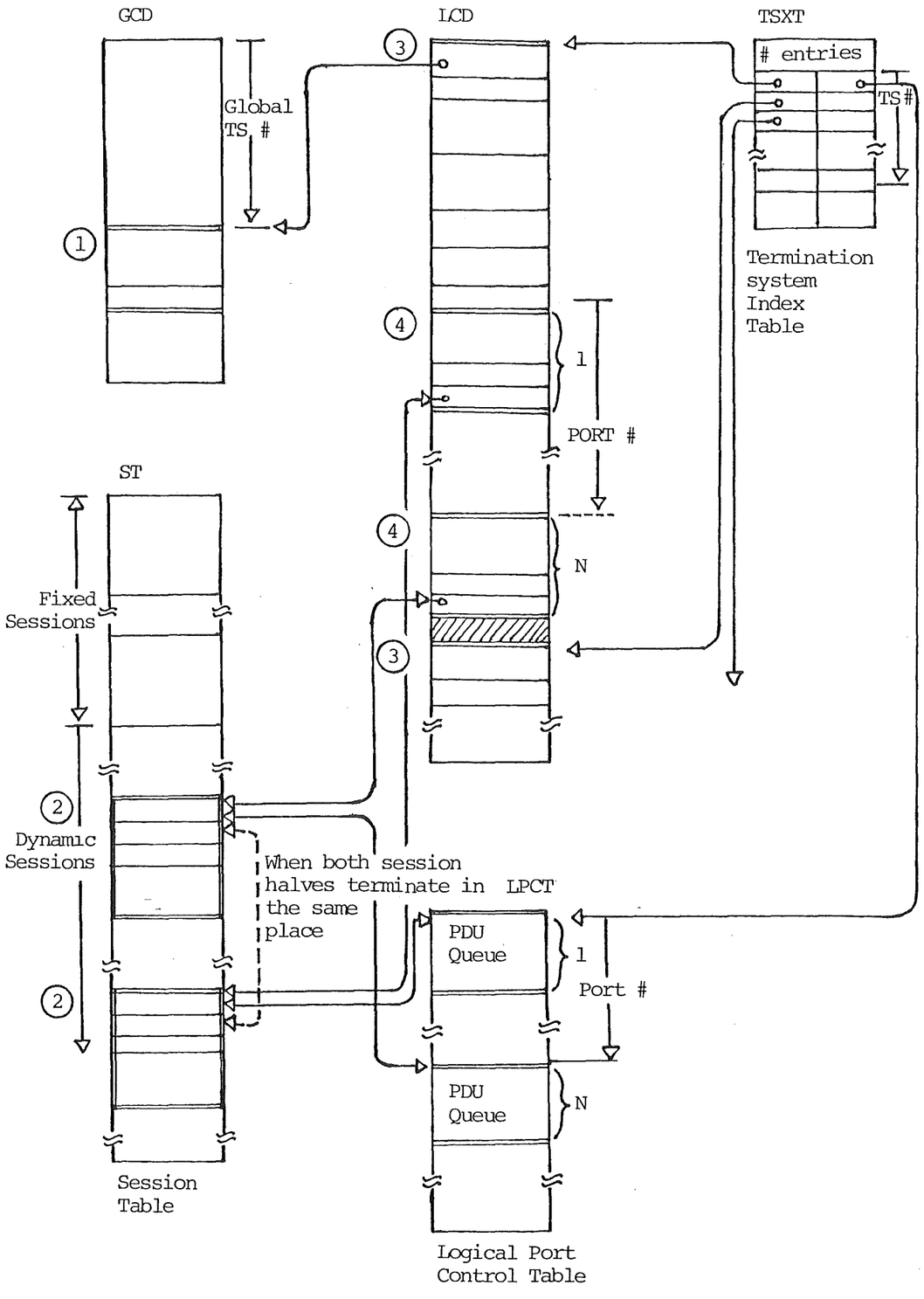


FIG. 5.2 SESSION CONTROL TABLES

(1) GCD ENTRY

Termination System Name
Node Address

(2) ST ENTRY

TS#, Port #
Node, Session
Flags
NDU Queue

(3) LCD TS ENTRY

Global TS #
Status
Incoming TS access rights
Outgoing TS access rights
Port Characteristics
Default Session
Ports

(4) LCD PORT ENTRY

Port Name
Status
Session #

- The capabilities of this TS in respect to each other TS within the network. The basic capabilities are:
  - Allowed to initiate calls to the TS
  - Allowed to accept calls from the TS
- Logical port characteristics
- Session establishment parameters

For each logical port within the termination system the LCD contains:

- The symbolic name of the port
- The port status
- The corresponding session number, if the port is active

5.2.1.3 Session Table

The Session Table contains the dynamic control information for each session half which originates or terminates in the node. If both end-points of a session are in the same node, there will be two Session Table entries, one for each port.

Each Session Table entry contains:

- The TS and port numbers of the corresponding logical port.

- The node number and session number used to address the alternate logical port.

- The address of the corresponding transport network circuit over which data is transferred.

- Input and output queues of Network Data Units (NDU's), corresponding to partially transferred Port Data Units (PDU's).

- Status information needed to control the flow of data over the session.

5.2.2 Session Control Functions

Two types of sessions exist between session control entities.

- *Fixed* (or *System*) sessions are statically configured at system generation time between every session control entity. Hence, if there are n nodes, session numbers $1..n$ are used by each session control entity to communicate with the other session control entities in the network.

- *Dynamic* (or *User*) sessions are established between logical ports by the session control entities at user request. The session control entities are analogous to telephone operators in exchanges, who use an inter-exchange link (the system session) to establish a connection (user session) between subscribers (logical ports).

Session control provides two main functions.

- *Session Management* services implement dynamic user sessions, providing a basic data transfer facility between logical ports.

- *Port Flow Control* services manage the transfer of information between logical ports over the session, providing user-oriented facilities.

5.3 SESSION MANAGEMENT SERVICES

Session management services establish the logical connections, and ensure reliable information interchange, between end-points anywhere within the network. This is achieved by the Session Transport Protocol, its interface to the lower level Transport Network services, and the Session Control Protocol.

5.3.1 Session Transport Protocol

The *Session Transport Protocol* (STP) is a packet protocol for data exchange between session control entities. This section describes the functions, formats and procedures of the STP.

5.3.1.1 STP Functions

While some STP functions may overlap with those provided by the transport network, the STP is the primary method of isolating session control services from dependencies on any particular data transport network characteristics. In particular, the STP provides independence in the areas of:

- Session Establishment

  Transport networks may either use a special circuit,

or a special packet type, to set up virtual circuits. Session Control provides both a fixed session between session entities, and a special packet type, to establish a connection between session end-points.

- Session Addressing

Session end-points are addressed by the session number within the STP header. The session numbers are allocated independently during the session establishment phase by the session control entities independently of any logical circuit mechanism provided by the transport network.

- Session Multiplexing

Since session end-point addressing is implemented at the STP level, multiple sessions between a pair of nodes can be multiplexed onto a single transport network circuit, or can occupy individual logical circuits, transparently.

- Session Error Control

Most local networks implement error control at the link level. The STP implements end-to-end error control, providing protection against failures of the underlying transport network services.

- Session Flow Control

Flow control mechanisms are required to prevent congestion within the network. In most local networks, flow control is limited to ensuring data integrity

at the link level. Although the X.25 packet level implements flow control for each virtual circuit, this is in the context of the local DTE-DCE interface rather than end-to-end between DTE's. The STP provides end-to-end flow control, or pacing, over sessions to prevent deadlock situations arising from congestion.

5.3.1.2 STP Packet Formats

The packet formats for the STP, illustrated in Fig. 5.3, are loosely based on the X.25 frame and packet levels, but have been adapted for the session control environment. The two significant changes are:

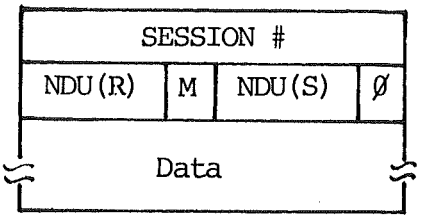
- The X.25 'more data' bit has been augumented by a more general flow control mechanism, required to avoid possible deadlock situations.

- The session establishment facilities are provided by a separate Session Control Protocol (SCP) which is embedded within the STP packets.

5.3.1.3 STP Procedures

The Port Data Unit (PDU) is the logical unit of data which is transferred between logical ports over a session. Since PDU's may vary in size considerably, a PDU which is to be sent is segmented into smaller limited-size Network Data Units (NDU's) as it is requeued from the logical port queue to the session table entry transmit queue. All but the final NDU have the *'more data'* bit set to indicate an incomplete PDU. As NDU's are received, they are queued to the session table entry receive queue in order while the 'more data' bit remains set. When the final NDU is received

DATA PACKET

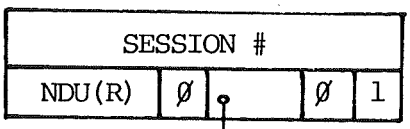


NDU(R) Next NDU # expected

NDU(S) NDU Send Sequence #

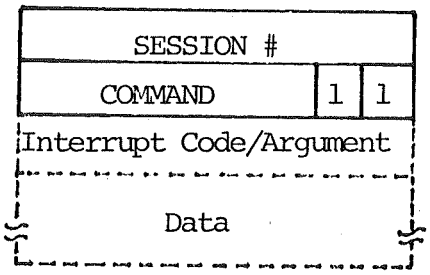
M More data follows - incomplete PDU

FLOW CONTROL



- 00 RR Receive Ready
- 01 RNR Receive Not Ready
- 10 REJ Reject
- 11 SW Set Window

CONTROL



Code	Command Type	Argument/Data
0000	Command Ack	Command to be acknowledged
0001	Interrupt	Interrupt code
0010	Interrupt Ack	Interrupt code
0011	Call Request	SCP parameters
0100	Call Acceptance	
0101	Clear Request	
0111	Clear Confirmation	
1000	Pacing Request	# NDU's to send
1001	Pacing Response	# NDU's allowed to send

FIG. 5.3 STP HEADER FORMATS

the NDU's are re-linked into a complete PDU which is passed to the destination logical port.

Care must be taken that the number of partially assembled PDU's do not exhaust the buffering capacity of a node and result in a 'reassembly lock-up' situation because insufficient buffers are available to complete the assembly of any one of the PDU's.

To ensure a smooth flow for small PDU's, each session direction establishes a *credit limit* determining, and reserving buffers for, the maximum number of NDU segments that it will accept without special arrangement. Providing that the PDU size is within the destination's credit limit, the corresponding NDU chain may be sent immediately. However, if the PDU size exceeds the credit limit, a special 'overdraft' or *pacing request* must be sent to the destination to ensure that sufficient buffer space is available. The entire PDU will be held until a *pacing response* is received approving the 'overdraft'. The arrangement applies only to that particular PDU; the normal credit limit applies for subsequent transfers unless another pacing request is made.

5.3.2 Transport Network Interface

Session Control is largely independent of the actual characteristics of the Transport Network (TN), since the TN protocols are 'wrapped' by the Session Transport Protocol which provides session addressing, flow control and error recovery. The TN is required simply to transfer blocks of data between nodes. This section defines the interface between the STP functions and the lower TN level.

5.3.2.1 Transport Network Functions

The *minimal* required function of the transport network is a simple *datagram* service between nodes: either to transfer a block of data to a specified node without change, or not to deliver it at all. It is not necessary that the blocks be delivered in the same order in which they were sent, although the session transport functions are simplified if sequentiality is guaranteed (as it is for most local networks).

It is assumed that the block length is in the order of 128 bytes or that, if significantly less (as in MININET; Section 3.3.2), it is possible to perform segmentation and reassembly at the transport network level transparently to the STP-TN interface.

Although these are the only functions required, sufficient information is provided by the interface to enable the full utilisation of higher level facilities, such as virtual circuits, if they are provided.

5.3.2.2 TN Interface Routines

The transport network interface is greatly simiplified because the transport facilities are 'wrapped' by the STP with only an underlying datagram service being assumed. Interface functions are required to pass STP packets to, and received STP packets from, the transport services. The only additional information required is the nodal address to which the packet is to be sent, or was received from. However, to maximise the independence between Session Control and the underlying transport services, additional redundant routines and parameters are provided to open and close circuits to a specified node. The routine to open a circuit returns a

'sub-channel' address which may be used by subsequent routines to identify a sub-channel or virtual circuit, if implemented. Sub-channel zero (0) is used by the fixed system sessions between nodes. Each routine also has an additional status parameter which indicates normal or abnormal completion of the operation.

The routines are entered at fixed locations in memory, which contain branches to the actual routine bodies. The routines to open and close circuits, and to send packets, are implemented externally to, but invoked by, the Session Control software. However, the routine invoked by the transport services on receipt of a packet is implemented within Session Control.

The particular interface routines are:

- OPEN (NODE, CHANNEL, STATUS, CALLER)

  The OPEN routine is invoked by Session Control to initiate a transport network circuit. The parameters are:

  NODE — The node to which the circuit is to be established.

  CHANNEL — The sub-channel number allocated and returned by the transport network services, which is referenced on subsequent operations. This is initially set to the session number by Session Control, but may be altered within the routine.

  STATUS — The status returned by the function indicates normal completion or an error.

CALLER Indicates whether the caller is the session source or destination node.

- CLOSE (NODE, CHANNEL, STATUS)

The CLOSE routine is invoked by Session Control to terminate a transport network circuit.

- SEND (NODE, CHANNEL, STATUS, PACKET, LENGTH)

The SEND routine is invoked by Session Control to send a packet to a node over a specified sub-channel. The parameters, other than those previously specified, are:

PACKET The address of the buffer which is to be sent.

LENGTH The length (in bytes) of the buffer.

- RECV (NODE, CHANNEL, STATUS, PACKET, LENGTH)

The RECV routine is a Session Control routine which is invoked by the transport network services level on receipt of a packet from a distant node. The routine passes the packet to Session Control for subsequent processing.

5.3.3 Session Control Protocol

The Session Control Protocol (SCP) is used to establish the logical connections between end-users. The three phases of session establishment within Session Control are:

- Interchanging control information between the source and destination Session Control entities using SCP-encoded packets.

- Establishing the transport network circuit for the session.

- Ensuring the network integrity by an initial exchange over the circuit.

5.3.3.1 SCP Functions and Formats

While the session data transport functions are well-defined, the control functions are virtually open-ended. There may be no uniformity even in functions which are common to all nodes, such as symbolic port-naming conventions. Furthermore, the specification of session facilities may range from a simple pairing of logical ports to a large number of session attributes. Therefore it is inappropriate to implement a fixed-format packet protocol, but rather an extensible mechanism is required which describes the data actually present. This is achieved by using the Item Descriptor (ID) protocol format described in Section 5.1.

ID-encoded strings are embedded within STP control packets which specify the function to be performed. Since each ID specifies an action or a parameter format, session establishment parameters may be encoded essentially in free format, providing related parameters are grouped together. For session establishment the minimal parameters are the initiating and destination Termination System and Logical Port names, and the session number or communication path to be used.

5.3.3.2 Session Establishment Sequence

Session establishment is initiated when a control message is received from a logical port. The message is analysed to determine the message type and ensure correct syntax, and encoded in ID-format for subsequent processing. The processing of a CALL message requires the following steps:

- If a session is not already active for the port, and the originating TS has access rights to the destination TS, a Session Table entry is allocated for this direction (half-session) and a CALL REQUEST packet containing the SCP-encoded parameters sent to the destination node's, Session Control.

- The destination Session Control ensures that the specified port exists and is not already active, and that the originator has access rights. If any of these checks fail, a CLEAR REQUEST is returned with the appropriate diagnostic code. Otherwise, a Session Table entry is allocated for the reverse half-session, a transport network services circuit initiated by calling the OPEN routine, and the destination's session number returned to the originator in a CALL ACCEPTANCE packet.

- If the establishment request was rejected, the originating Session Control reports a diagnostic message to the originator and releases the Session Table entry. Otherwise, a transport network services circuit is initiated with the OPEN routine and the Session Table entry updated to contain the circuit number, and the session number returned by the CALL ACCEPTANCE packet.

- A SET WINDOW command is sent from the originator to the destination and returned over the newly-created circuit. This both confirms the integrity of the circuit and initiates the data transfer phase.

5.4 LOGICAL PORT SERVICES

Once the session has been established, the end users may communicate across the session. The STP controls the flow, sequencing and integrity of the internal Network Data Units (NDUs), providing reliable data transportation between paired session control entities regardless of where they are located in the network. However, further functions are required to control data flow between the logical ports, and to interface between the logical ports and the session transport facilities. These functions are provided by the Port Flow Control (PFC) protocol and the Logical Port Multiplexor (LPM) component of Session Control.

5.4.1 Port Flow Control Protocol

Port Flow Control regulates the interchange of messages, known as Port Data Units (PDUs), between logical ports. It also provides basic facilities which are used by the Data Flow Control component of the virtualisation level to implement various terminal interaction modes.

5.4.1.1 PFC Functions

The PFC protocol is required to implement pacing, end-to-end acknowledgement and flow control functions between session end-points, rather than the data integrity functions

emphasised by lower levels.

The send-receive mode selected at session establishment time determines the message flow during the session. This message flow is completely independent of the flow at the transport level, being solely determined by the requirements of the user interface. Three session modes which are defined are:

- Two-Way Simultaneous (TWS)

  In TWS mode, transfers proceed in both directions simultaneously and independently of each other. Responses and interrupts are expedited, so that they have priority over normal data transfers.

- Two-Way Alternate (TWA)

  In TWA mode the two ports take turns at being active. The port which is active can permit the other to assume the active role by sending a 'change direction' indicator. TWA mode is a natural conversational mode for terminal interaction. The mode controls only normal data interchange - responses and interrupts are transmitted without restriction, as in TWS mode.

- Two-Way Contention (TWC)

  In TWC mode, the port at either end may initiate a transfer. If the partner does not simultaneously want to send, the initiator retains the active role until it is relinquished. At this time both ports are back in contention mode again. Contention is always resolved in favour of one party, determined at session establishment.

The normal mode of operation is TWS or TWA. Contention mode is provided for terminal interactions, where output and input operations for a device must exclude each other, but there is no regular alternating conversation flow between the end points.

Paired logical ports can be considered to be directly connected by a queue of PDUs in each direction, without regard to lower levels. Pacing is concerned with regulating the length of this queue, both to ensure a smooth flow of data between ports and to prevent overloading of the network.

In practice, there is little advantage to be gained from allowing more than two PDUs in transit between paired ports at any time, since this is sufficient to allow the overlap of acknowledgements with data transfers when device speed limits the overall throughput rate. For terminal interactions, often only a single item is transferred alternately in each direction. Pacing is controlled by a window mechanism using the PDU(R) and PDU(S) sequence numbers, similar to the STP flow control.

End-to-end acknowledgement may also be provided by the PDU sequence numbers if each PDU is acknowledged only after it has been processed by the destination port, rather than when it is queued to the destination port.

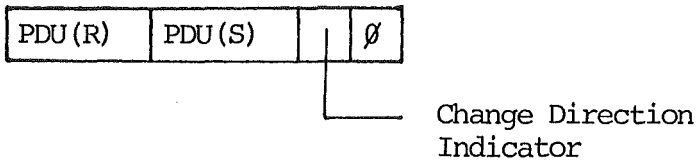
Flow control functions are required to suspend and resume data flow over a half-session between paired ports, and to discard data traffic over a half-session until further notice.

5.4.1.2 PFC Header Formats

A single-byte PFC header, illustrated in Fig. 5.4, is inserted at the beginning of every PDU, or may be sent in isolation for control purposes. Each PDU header contains the Send sequence number PDU(S) of this PDU, and the expected Receive sequence number PDU(R) of the next PDU to be received in the alternate direction. Control PFC headers contain a command function and a parameter, normally PDU(R).

The PDU headers defined for Port Flow Control are:

SUSPEND	Indicates a 'receive not ready' (RNR) condition at the originating port. On receipt, sending of further PDUs to the port will be inhibited until receipt of a RESUME, or a data packet with the CDI bit set.
RESUME	Indicates the lifting of a previous SUSPEND condition, or the termination of a FLUSH sequence. PDUs may now be transferred normally.
FLUSH	Indicates that queued and all subsequent PDUs should be discarded at the source until a RESUME is received.
RESYNC	A response to a FLUSH which indicates the next PDU number to be sent.
ACKNOWLEDGE	Updates the lower edge of the receiving port's transmit window. If the port was blocked on the maximum PDU window, transmission may resume.



DATA

000	Suspend
001	Resume
010	Flush
011	Resync
100	Acknowledge
101	Solicit Status
110	Set Window

CONTROL

FIG. 5.4 PFC HEADER FORMATS

SOLICIT STATUS	Sent periodically when this direction of the session is closed and PDUs are queued to be sent.
SET WINDOW	Indicates the PDU pacing window size for the port.

5.4.2 Logical Port Multiplexor

The Termination System - Session Control (TS-SC) interface is the real connection between the individual Termination System port and the transport facilities. As such, the TS-SC interface is the primary point at which data flow through the network can be regulated. The Logical Port Multiplexor (LPM) builds the PFC headers and implements the interface between each logical port and the session management and transport services.

5.4.2.1 Interface Functions

Once Session Control has accepted a PDU, it is committed to deliver it to the destination node. The TS-SC interface may be closed (and PDU's queued) for any of the following reasons:-

- Session Control is temporarily not accepting further PDU's because the session credit limit has been reached, or a Receive-Not-Ready (RNR) has been received from the destination.

- The remote port has indicated that it cannot accept further data at this time by sending a SUSPEND.

- An expedited signal is to be sent, or a response to a signal is outstanding.

- The PFC Pacing (credit) window limit has been reached.

5.5 CONFIGURATION SPECIFICATION

The configuration specification defines the network structure in terms of:

- The *occurences* of entities which comprise the network,
- The *attributes* of each entity,
- The *relationships* between the entities.

The network structure is described from the viewpoints of:

- The logical configuration of each node,
- The physical configuration of each node,

  and

- The global network configuration of nodes.

Session Control is primarily concerned with the logical structure which defines the contents of the session control tables described in Section 5.2, and the global network structure for addressing and routing. The physical configuration specification is used to select the required operating system functions, and to distribute tasks between processors in a multi-processor environment.

5.5.1 Syntax Structure

A standard 'syntax skeleton' has been developed for all configuration statements. This provides a number of advantages, including:

- A consistent language structure for the user,
- Simplified implementation, since table-driven parsing techniques can be used,

  and

- Easy enhancement, since new entities, attribute types or attribute values may be added by simply amending the tables.

Each configuration statement has three mandatory components, illustrated in Fig. 5.5. These are:

- The *entity name*, a logical name or label by which the entity is referred to,
- The *entity type*, determining the type of entity being defined,

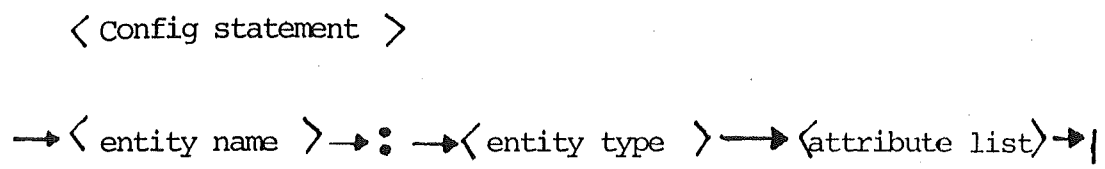
  and

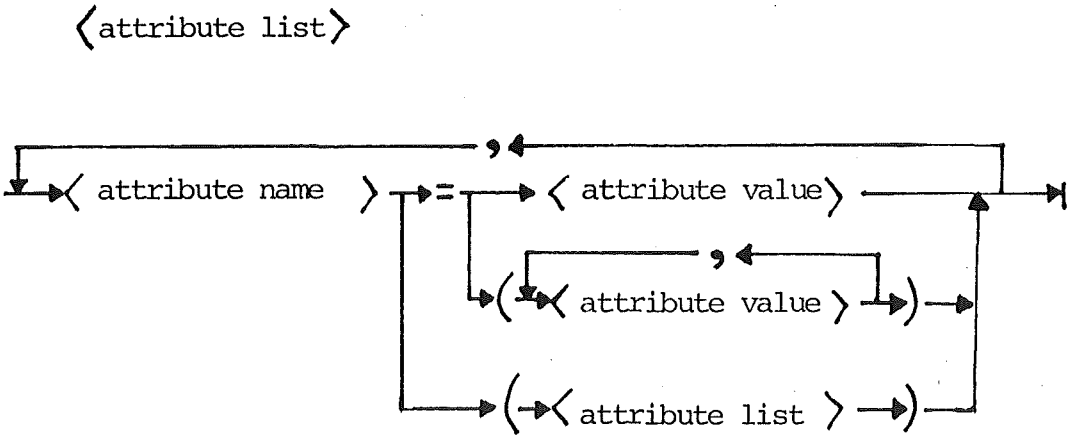
- The *entity attributes*, which describe the characteristics of the entity and its relationships with other entities.

5.5.2 Logical Node Configuration

The logical node configuration describes the structure of each node as seen by session control in that node. It is

SYNTAX





EXAMPLE

```
L1:     LINE    TYPE=(SYNC, SIMPLEX), CHARSZ=8,
                PARITY=NONE, MODEM=BELL2Ø1C,
                PROTOCOL=BSC, PROC=P1, ADDR=X'FFØØ';
```

FIG. 5.5 CONFIGURATION STATEMENT SYNTAX

used to build the Local Configuration Directory (LCD), described in Section 5.2.1.2.

Each node is addressed as a hierarchy of Termination Systems (TS's) and logical ports (PORTs).

5.5.2.1 NODE Statement

A *Node* is the highest level entity in the network, corresponding to a physical network computer. Each NODE statement must specify the node name, and the network address used to identify the node for routing purposes.

5.5.2.2 TS Statement

A *Termination System* (TS) identifies a logically related group of ports within the node. The ports may be related by the external interface, or by the capabilities that they are allowed. For example, TS's may have limited access rights to other TS's within the network.

Each TS statement must specify a unique name for the TS, and the default PFC mode and window size for the ports. The TS statement may also supply optional session-establishment related parameters.

5.5.2.3 PORT Statement

A *Port* identifies an addressable logical end-point with which a session may be established. It may correspond to a physical terminal device, a user on a host computer, or a virtual circuit of an external communications system or network.

```
%
%          LOGICAL CONFIGURATION
%
N1:        NODE        NODEID='NODE1',ADDR=X'Ø1';

TS1:       TS          TSID='TS1',MODE=TWA,WINDOW=1;

TS1P1:     PORT        PORTID='PORT11';
TS1P2:     PORT        PORTID='PORT12';
TS1P3:     PORT        PORTID='PORT13';

TS2:       TS          TSID='TS2',MODES(DEFAULT=TWA,OPT=
                       (TWS,TWC)),WINDOW=2;

TS2P1:     PORT        PORTID='PORT21';
TS2P2:     PORT        PORTID='PORT22';
%
%          PHYSICAL NODE CONFIGURATION
%
N1P1:      PROC        TYPE=8080,
                       ROM(SIZE=8,ADDR=X'0000'),
                       RAM(SIZE=16, ADDR=X'2000');
                       TS=(TS1,TS2);

BSCL:      LCLASS      TYPE=(SYNC,SIMPLEX),CHARSZ=8,PARITY=NONE,
                       MODEM=BELL201C,PROTOCOL=BSC;
BSCT:      TCLASS      LCLASS=BSCL,DEVICE=D2780;

N1L1:      LINE        LCLASS=BSCL,ADDR=X'FFØØ',PROC=N1P1;

L1T1:      TERM        TCLASS=BSCT,PORT=TS1P1,ADDR=X'1330';
L1T2:      TERM        TCLASS=BSCT,PORT=TS1P2,ADDR=X'1331';
L1T3:      TERM        TCLASS=BSCT,PORT=TS1P3,ADDR=X'1332';
L1T4:      TERM        TCLASS=BSCT,PORT=TS2P1,ADDR=X'1333';
L1T5:      TERM        TCLASS=BSCT,PORT=TS2P2,ADDR=X'1334';
%
%          GLOBAL NETWORK CONFIGURATION
%
T12:       TRUNK       NODES=(N1,N2);
T13:       TRUNK       NODES=(N1,N3);
T23:       TRUNK       NODES=(N2,N3);
```

FIG. 5.6 EXAMPLE CONFIGURATION SPECIFICATION

Each PORT statement must specify a symbolic port name (unique within the TS) by which the port can be addressed during session establishment.

5.5.3 Physical Node Configuration

The physical node configuration specifies the structure of each node in terms of its physical components and their relationships to the logical structure. It primarily influences the code generated for a node, determining:

- The processor structure and functional distribution of the node,
- The types of communications hardware present and the corresponding operating system functions which must be provided for their support,

  and

- The basic device classes which must be supported.

Each node is described by a hierarchy of processors, lines and terminal devices. Detailed discussion of the line and term attributes is deferred to the following chapters.

5.5.3.1 PROC Statement

PROC statements are used to specify the characteristics of each network computer, and the distribution of logical functions to processors.

Each PROC statement must specify the microprocessor type (currently the 8080 is supported for general tasks, and the

8741 for line protocol related functions) and the type and disposition of its memory. Every TS within a node must be associated with a processor.

5.5.3.2 LCLASS and LINE Statements

These statements are used to specify the hardware attributes of communications links. The LCLASS statement defines a default line 'profile', while the LINE statement specifies a specific hardware interface and relates it to a processor.

5.5.3.3 TCLASS and TERM Statements

The TCLASS statement defines a default terminal profile for a class of terminal devices.

The TERM statement relates a specific line and device type to a logical port, providing the link between the internal network and the external access interface.

5.5.4 Global Network Configuration

The global network configuration specifies the connectivity between nodes which is used to build a routing matrix for mesh (non-bus) architectures, in which it is not possible for every pair of nodes to communicate directly without requiring the intervention of intermediate nodes.

5.5.4.1 TRUNK Statement

The TRUNK statement defines a communication path between two (or more) nodes. For a simple broadcast or ring network

structure, a single TRUNK statement may be used to define a connection between all the nodes.

It would be possible to implement the transport network functions entirely within the Network Interface Machine structure if each trunk was associated with a corresponding line definition between the nodes.

6. DATA PRESENTATION SERVICES

Lower network levels have been concerned with the transfer of data unchanged from source to destination. However, there is rarely complete agreement in the conventions adopted at both ends, unless specifically designed to work together. Often there may be significant differences, which will become more acutely felt when a large amount of diverse equipment is interconnected. Data presentation services are concerned with the systematic implementation of the transformations required between a data source and its destination. In the most general case, they could apply to any kind of data transmitted - text, command languages, file records and even programs. A network in which any program could be executed on any type of machine (within physical limitations) is not unreasonable in view of current 'portable' compiler technology, and many manufacturers have developed automated file conversion packages for inter-machine conversions. This chapter is concerned specifically with the transformations required to format text for particular devices; however the techiques used have more general application.

The application of data presentation services to insulate users from unnecessary device dependencies is commonly known as *terminal virtualisation*. This chapter examines terminal characteristics and the facilities required by users to define a set of terminal profiles *(virtual terminals)*, each representing a class of devices. From these the functional structure of the virtualiser is developed, the Data Presentation Protocol (DPP) and Data Flow Control protocol (DFC) defined, and the terminal attribute specification language described.

6.1 DEVICE HARDWARE DEPENDENCIES

A device's hardware characteristics influence the transmission interface and particular functions of the device itself. The electrical characteristics of the interface are well standardised, but a device's hardware constraints may influence procedures over the interface.

6.1.1 Transmission Interface

Bits of binary data are transferred between a computer and a terminal by changes in voltage (RS232-C interface) or current (current loop interface). Data may be transferred in parallel over several lines simultaneously or, more commonly, bit by bit serially over a single line. The transfers may be *synchronous*, in which successive characters are transmitted in a continuous stream and isolated by position, or *asynchronous*, in which each character is preceded by a 'start bit' and followed by a 'stop bit' and may be transmitted at a non-uniform rate. The transfer rate generally varies from 100 to 9600 baud (bits per second). Most terminals use the 7-bit ASCII character code with an extra bit padding the character to 8 bits. This additional bit is often used for parity error detection.

The electrical interface generally poses few compatibility problems. Most asynchronous terminals are switch-selectable for combinations of speed and character formats. Synchronous terminals may also be connected easily, although they require special processing to implement the line protocol. The only precaution which must be observed is to ensure dissimilar (active and passive) current loop interfaces are interconnected.

6.1.2 Device Functions

Many device control functions have hardware dependencies which must be provided for in the interface software. These are evident in timing restrictions and control-function implementations.

6.1.2.1 Timing Restrictions

Many device control functions require significantly more time to complete than the transmission period of the invoking character, particularly at higher data rates. For example, it takes much longer to perform a carriage-return operation on a printer than to print a single character, although both functions are invoked by a single character code. Even a completely electronic device, such as a Visual Display Unit (VDU), may take a significant time to erase or scroll the screen.

Many devices are unable to process further data during this time, so transmission must effectively be stalled while the operation is completing. The particular characters requiring delays and the delay periods vary from device to device. These delays may be implemented by inserting an appropriate number of 'pad' characters (which are ignored by the device) following the function or, with asynchronous devices only, pausing data transfer for the required period. Some asynchronous devices with internal buffering are able to signal the sender to cease transmission during execution of a control function and, on completion, to resume by sending special control codes to the computer. The ASCII codes used are known as XOFF and XON respectively.

6.1.2.2 Control Function Implementation

Significant differences are encountered between devices in the interpretation of control functions. The basic device control functions are known by the mnemonics BS, HT, LF, FF and CR, and in ASCII are represented by the octal values 010 through 015. The major differences are in the following areas:

- Line-Feed/New line

  Some devices interpret the Line Feed (LF) character to imply only vertical motion of the current printing position, and cause the Return key to generate a single Carriage Return (CR) character. Others interpret the Line Feed to imply movement to the first position of the following line, and cause the Return key to generate the sequence CR LF. Incompatibility will result in overprinting or double spacing of output.

- Horizontal Tabulation

  Some devices interpret the Horizontal Tabulation (HT) character to advance to the next predetermined tabulation position (often a multiple of 8 characters), while others ignore it or regard it as a space character.

- Vertical Tabulation/Forms Control

  Some printing devices include hardware to automatically advance a number of lines on receipt of a Vertical Tab (VT) character, or to the top of the next page for a Form Feed (FF) character. The tabulation positions and page height may be selectable by interchanging control bands, or by switch settings. Other devices simply

interpret both as new-line functions.

- Folding

  Folding refers to the treatment of output which exceeds the physical printing width of the output device. When folding is implemented, the excess output is continued onto the next line; otherwise the output is truncated at the maximum width.

6.2 PROGRAM INTERFACE

The program interface is concerned with those characteristics of the device which affect a program communicating with it. These include the page format, displayable character set, device capabilities, and the character sequences required to invoke device control functions.

6.2.1 Display Format

The display format typically varies in depth from 12 lines to 66 or 88 lines, and in width from 64 to 132 columns. The most common sizes are 24 lines by 80 columns for VDUs, and 66 lines by 80 or 132 columns for printers. A VDU screen image may be displayed in any order, while a printer page must be produced sequentially line by line. Output must be formatted within the constraints of the particular device, or transformations applied to map the logical page onto the physical page size.

6.2.2 Character Set

ASCII is the most universal character set. However, many devices, particularly printers, only implement a subset of the possible characters. The lower case alphabetics are

commonly omitted, along with some of the less commonly used special characters. In addition, devices may provide special alternative character sets for graphics or special languages (eg. APL or the Japanese Katakana character set).

6.2.3 Device Capabilities

Device capabilities may vary from the simplest printing devices, which sequentially output lines of text with no recognition of page-oriented functions, to visual display units which dynamically control the entry and display of data over the entire screen image.

The basic device capabilities are:

- Character/Block Mode

  On input, devices may operate either in character or block mode.

  - In *character mode*, each character is passed over the communications interface as the key is depressed. No internal processing is performed by the terminal and the computer must perform all input editing functions.

  - In *block mode*, each character is buffered internally within the terminal until a message (which may be up to an entire screen of data) has been assembled. The message may be edited locally within the terminal, then transmitted in a single block to the computer.

- Data Entry Control

  As the computer has no direct control over data as it is entered with block mode terminals, they generally provide facilities to define specific data entry fields on the screen.

  The simplest and most common facility is known as *protected fields*. Using control sequences, the screen can be formatted into protected (read-only) areas of text, and unprotected data entry fields. Data can only be entered into unprotected areas of the screen, and at the end of each entry the cursor automatically advances to the next field. Cursor positioning keys and the tab keys may also be used to step between the fields.

  More sophisticated data entry terminals may also permit the specification of data field attributes, such as data validation criteria, justification, and entry requirements.

- Character Formats

  Many of the more sophisticated devices offer a variety of fonts or highlighting features which may be used to control the way in which the data is displayed. Selectable fonts may include bold face or large-size characters. The most common attributes are:

  - Reverse (negative) image
  - High intensity
  - Blinking
  - Underscore

6.2.4 Control Sequences

The basic printer-oriented device control functions are invoked by a set of reserved characters (Section 6.1.2.2). Rather than introducing an increasing number of control characters to implement special functions for VDU terminals, control sequences were developed in which a leading *function shift* code caused the following data to be re-interpreted as control parameters rather than normal data. The Escape (ESC) character (octal code 033 in ASCII) is the most commonly used function-shift character.

Initially each manufacturer developed private standards for device control sequences which, although ensuring compatibility within a product line, resulted in a general incompatibility between different manufacturers' devices. The differences in control sequences and their interpretation for some popular terminals is illustrated in Fig. 6.1. Some terminals are able to emulate a number of different terminal protocols, as well as the ANSI Standard control sequences. The ANSI Standards (Fig. 6.1), which have only recently been developed, will result in an increasing compatibility of future terminals.

6.3 USER FACILITIES

Effective user interaction with a computer requires more than simply being able to enter and output data. A well designed user interface will allow the user to work in the most comfortable and natural manner, rather than draw attention to its own shortcomings, while a poorly designed user interface will cause constant frustration. Some systems are enthusiastically accepted by their users, while others never are; this may be largely due to the interface perceived by

FUNCTION	DEC VT52	BEEHIVE B150	HAZELTINE H1500	ANSI
LEAD-In	ESC 0'33'	ESC	~0'176'	ESC [
CURSOR UP	ESC A	ESC A	~$\hat{R}$	ESC [A
DOWN	ESC B	ESC B	~$\hat{L}$	ESC [B
RIGHT	ESC C	ESC C	~$\hat{K}$	ESC [C
LEFT	ESC D	ESC D	~D	ESC [D
wrap	No	Yes	No	No
POSITION	ESC Y Line Col	ESC F Line Col	~DCl Col Line	ESC [line; Col H
coords	31 + coord	31 + coord	co-ord	ASCII string
ERASE to EOL	ESC K	ESC K	~K	ESC [OK
from start				ESC [1K
line				ESC [2K
ERASE to EOS	ESC J	ESC J	~J	ESC [0J
from home				ESC [1J
display				ESC [2J

FIG. 6.1 TERMINAL CONTROL SEQUENCES

the user rather than an inherent property of the system itself.

This section examines the basic facilities required for effective man-machine interaction using a terminal device.

6.3.1 Input Facilities

Block mode terminals contain all the functions required to format input text within the terminal itself. The following facilities must be provided for character mode terminals which possess no in-built data processing capabilities:

- Echoing

  A terminal consists of an input device (keyboard) and an output device (printer or display) which, although physically united, are logically quite separate. As characters are entered on the input device they are normally displayed simultaneously (echoed) on the output device. This may be accomplished by a direct linkage within the terminal, or by the computer returning each character as it is received to the output device. Computer generated echo has the advantages that it both confirms that the characters have been received correctly, and allows special processing to be performed for control functions.

  When a data communications network is interposed between the terminal and the computer, it is generally not feasible to allow communication on an individual character basis. Because network overheads are incurred largely on a packet basis,

  - The user would experience the round-trip delay in echoing each character transmitted,

- The requirement for source-device related functions at the destination would result in a loss of device transparency across the network.

For these reasons, echoing should be performed at the terminal-network interface. This may require a corresponding suppression of echo at the destination to avoid repetition. When special conditions prevail, a character by character 'escape' mode can provide complete network transparency.

- Editing

As previously mentioned, block mode terminals contain editing functions which allow complete formatting of messages internally before they leave the terminal. When each character is passed individually to the computer, the computer itself must provide editing facilities for the message being assembled.

The most basic editing function is that of deleting the last character of the current line. On a screen device this may be indicated by visibly erasing the character from the display (with proper allowance for tab characters). Printing devices pose a greater problem; the possibilities are;

- Echoing some visible representation of the 'delete' character, such as an underscore or cross-hatch (#),

- Echoing the erased characters (possibly bracketed by delimiters),

- Backspacing and overprinting the deleted characters with some high intensity (solid) symbol.

In all cases, if the input buffer is empty the deletion should be ignored with no visible effect. When the edited input message cannot easily be displayed on the device, it should be possible to redisplay the current input at the user's request.

To provide the equivalent functions of block mode devices it is also desirable to allow the insertion, replacement and deletion of characters within the line through the use of control functions. On a screen device the display may be updated accordingly; on a printer it would be necessary to retype the amended text on a new line.

It should also be possible to abort the current input line entirely, accompanied either by erasure of the line on a display, or the echoing of some suitable character string to a printer.

- Delimiting

When input characters are echoed and assembled into a message before being passed across the network, there must be a recognisable indication of the end of the message. Normally this will be the Return key, although some special programs such as editors take action on punctuation or special control characters. These characters are known as *wake* characters since they initiate action at the destination. Usually these characters will also *break* or inhibit input until a response is received from the destination. The user should be provided with sensible defaults, such as the Return key, but also be able to select other characters if required by the application. These techniques allow

network transport delays to be absorbed within the processing delay and hidden from the user.

- Function Keys

  Many terminals provide a set of 'function keys' which pass special codes immediately to the destination. Where necessary these should be simulated using special character sequences.

6.3.2 Output Facilities

The major output facilities required are control over the format and rate or presentation of output to the device.

Many programs are written on the assumption that the output device will be a printer with a page of 66 lines by 132 columns. When such output is directed to a terminal, problems arise due to the limited display size of the screen - usually 24 lines deep by 80 columns wide. Functions are required to map the logical page onto the physical device.

6.3.2.1 Height Restriction

The limited height of a VDU screen may be overcome by restricting the rate at which information is displayed. These methods are:

- Page Scrolling

  Page Scrolling refers to the suspending of output each time the screen becomes full, until the user requests further output. Each new screen image may erase the display and recommence at the top line, or may roll the previous page off the top of the screen

as new lines are added at the bottom. A logical page change within the output should also terminate the current screen page. It is desirable to be able to scroll the screen both forwards and backwards within the current logical page.

- Line Scrolling

  Line scrolling refers to 'holding' output and displaying each line only at the user's request. This allows the user to read the text as it is output.

- Variable Scrolling

  Variable scrolling refers to the suspension and resumption of output at any time through the use of control functions. On many asynchronous devices the Control-S and Control-Q keys will generate the XOFF and XON codes in the same way as the hardware buffer control functions. With high speed devices it may be desirable to limit the rate at which new lines are displayed.

6.3.2.2 Width Restrictions

When the logical line width exceeds the physical device width, some transformation must be applied to prevent loss of information.

- Folding

  Folding refers to the continuation of the excess text onto the following line. A continuation line should be indicated either by indenting or some special 'continuation' character string.

- Truncation

  Truncation refers to the discarding of a portion of over-length lines. It may be more appropriate to truncate on the left, rather than the right, if the output begins with non-significant information or leading spaces.

- Compression

  Compression refers to the removal of extraneous spaces from lines of output. In many cases the line width may be substantially reduced by compressing each sequence of spaces into a single space.

6.3.2.3 Output Redirecting

It should be possible to redirect output to an alternative destination, such as a printer shared among a cluster of VDUs. This printer must be dedicated to the terminal for the duration of each page, or a sequence of pages.

It should also be possible to discard output queued to a terminal without displaying it, and to resume display at the user's request. This is particularly useful when a large volume of output is directed to a slow device, much of which is not required.

6.3.3 Process Control

In addition to the input of data, it is also necessary to pass commands to the operating system. In many cases these will be normal messages, which are differentiated by the operating system through a reserved character sequence starting the line. However, in some cases, single-character

control functions may be used. To ensure general communicability it is necessary to map between these two conventions for:

- End-of-File signal

  The End-of-File signal indicates the end of a logical input phase. It is often represented by the Control-Z function on minicomputers.

- Interrupt signal

  The interrupt signal indicates that the process executing is to be interrupted, and usually terminated. It is often represented by the Control-C function.

6.4 VIRTUAL TERMINAL MODELS

In order to provide a general communicability between devices it is necessary to create common device models defining the capabilities of the devices. The term 'virtual' terminal is often used to decribe these models. This section examines the concepts and attributes of the virtual terminal models.

6.4.1 Virtual Terminal Concepts

The *Network Virtual Terminal* comprises a standard character code to represent all printing symbols and control functions, together with a complete definition of the actions that the control functions would have on the model terminal. A

virtual terminal handler implements the transformations required to ensure corresponding behaviour of the physical device with the terminal model.

A considerable degree of device independence may be obtained by thinking in terms of a Logical Presentation Space (or logical page) that is not tied to particular device characteristics. The LPS has a defined size, determined by the maximum line and column positions. The output image is obtained by applying the Virtual Terminal Protocol (VTP) specified actions to the LPS, and then transformations are applied to map the LPS onto the physical presentation space (printer page or screen display), using scrolling and width-reducing techniques where necessary.

The virtual terminal protocol must provide an encoding for all the required operations of the virtual terminal. These may be divided into the following groups:

- Position Control

  The position control group includes the static (parameterisation) commands which define attributes of the LPS, and the dynamic (movement) commands which control the current active position within the LPS at which the next data will be displayed.

- Format Control

  The format control group controls the attributes of the output text, such as font, underlining, etc.

- Edit Control

  The edit control group controls the visibility of data by screen-editing functions, such as erasure.

- Field Control

  The field control group determines the attributes of input data within particular portions of the LPS.

6.4.2 Virtual Terminal Profiles

The major problem with the concept of a virtual terminal is the mapping between the model and the physical device. This can be considered at two levels:

- Determining the overall facilities of the virtual terminal

- The detailed mapping between each facility and the actions of the physical device.

The philosophies regarding the capabilities of the virtual terminal are:

- Provide conversions where necessary so that any physical device can invoke all the facilities of the virtual terminal,

- Select a subset of facilities used which is compatible with the capabilities of both devices communicating,

  or

- Restrict the facilities to that subset possessed by the lowliest of terminals, and provide only a basic

communication facility.

The first is difficult (and may be impossible) to achieve since it is not possible to completely ignore the inherent physical limitations of particular devices, while the last is unduly restrictive, negating many of the advantages of a network. Rather than implement a selection of facilities on an individual device level, a solution is to recognise the existence of classes of devices, and to provide one model for each class.

Three models are proposed, together with an 'escape' mode which allows individual device capabilities to be fully exploited when necessary.

6.4.2.1 Scroll Mode

Scroll mode models the class of printing devices, being guaranteed only to interpret basic positioning commands. Because printing is inherently a sequential operation, a move to a line before the present will be interpreted as a move to the corresponding point on the next page.

The LPS is typically 66 lines deep by 132 columns wide; when the physical device image is smaller, reduction techniques must be applied.

Two formatting commands (underlining and boldface) may be used, but the particular interpretation (if any) is not defined. Both are most likely to be implemented by underlining on printing devices, or by boldface on display devices.

6.4.2.2 Display Mode

Display mode models the class of general visual display units. It interprets positioning commands without restriction. The LPS is typically 24 lines deep by 80 columns wide, corresponding directly to the physical display.

A complete set of display format attributes are specified; however not all may be implemented as specified on a particular device, but may be mapped onto the closest alternative.

Display mode devices may have block mode capabilities on input or, where specified, this capability may be emulated for character oriented devices by buffering the screen image in memory.

6.4.2.3 Data Entry Mode

Data entry mode models the class of intelligent terminals specially designed for commercial data entry operations. The terminals have inbuilt data editing and validation functions which are applied to specified fields in the display, specifying the valid data types, field type and justification.

6.4.2.4 Native Mode

Native mode provides a 'transparent' device interface by which raw data can be passed directly to the terminal without any formatting intervention. This allows special device capabilities, such as graphics, to be fully exploited by programs written for the particular device.

6.4.3 Virtual Terminal Implementation

The virtualisation functions are implemented at two different levels within the NIM. These are:

- Data Presentation Services

  Data Presentation Services (DPS) performs the translation between device-dependent and device-independent formats, and controls the interaction with the alternate session end point, using two protocols:

  - The Data Presentation Protocol (DPP), which encodes the LPS image for transfer across the session

  - The Data Flow Control (DFC) protocol, which implements the flow procedures controlling the interaction between the two session end points.

- The Device Access Facility

  The Device Access Facility (DAF) provides the communications interface between the device and the data presentation layer, emulating device functions as required.

The combination of the DPP and the DFC together make up the virtual terminal protocol.

6.4.3.1 Data Presentation Services

The Data Presentation Services (DPS) implement the transformation between the DPP functions and the corresponding control sequences of the physical device, including timing delays where necessary. There are three classes of DPS, for scroll, display and data-entry modes. Each DPS within a class provides the same overall functions for every device type, but performs specific transformations for each type of device.

There are two approaches to specifying the device-dependent processing; tables interpreted by generalised routines, or specific routines for each device.

Table look-up can effectively provide direct translations and access to quantitative device attributes; however it is difficult to describe complex transformations functions by tables without incurring significant interpretation overheads. For these functions, specific routines are much faster, and may not occupy significantly more space.

The approach taken is to implement a generalised presentation services program for each mode, and a table specifying device attributes and the addresses of device-specific transformation routines for each device type.

6.4.3.2 Device Access Facility

The Device Access Facility (DAF) provides the communications interface to the device.

For synchronous devices, a communications protocol handler is required to implement the particular line protocol. The implementation of communications interfaces is described in detail in the following chapter.

For character devices, message assembly and input editing functions are required, as described in Section 6.3.1.

6.5 DATA PRESENTATION PROTOCOL

The Data Presentation Protocol (DPP) provides a common network-wide encoding of the virtual terminal functions. One transformation may be performed between the source device and the DPP, and a different transformation between the DPP and the destination device, when the device types differ.

6.5.1 DPP Functions

The functions provided by the virtual terminal are a subset of the ANSI terminal control sequences standardised in documents X3.41-1974 and X3.64-1977. While some device may implement functions not included in this subset, it has been chosen to represent a 'kernel' of core functions provided by, or capable of emulation on, most popular terminals.

6.5.1.1 Data Transfer Group

The data transfer group effects the transfer of data between the source and destination logical presentation spaces. The functions defined are:

TXT *ASCII Text*

A sequence of displayable symbolic character codes.

ITS *Inter Text Space*

An encoded string of space characters, used for data compression.

FNK *Function Key*

A code corresponding to a terminal function key.

SIG *Signal*

An operating system end-of-file or interrupt notification.

6.5.1.2 Position Control Group

The position control group determines the placement of text within the LPS. Functions defined are:

FF *Form Feed*

Move to the next presentation surface.

LF *Line Feed*

Move down one line at the current column position.

CR *Carriage Return*

Move to the first position of the current line.

NL *New Line*

Move to the first position of the next line.

HVP *Horizontal Vertical Position*

Move to the specified line and column position.

TAB *Horizontal Tab*

Move to the next tab position on the line. If no further tab positions are defined, interpret as a single space.

HTS *Horizontal Tab Set*

Set a horizontal tab stop at the current position.

TBC *Tabulation Clear*

Clear the tab stop at the current position.

6.5.1.3 Edit Control Group

The edit control group alters the layout or positioning of previously entered or displayed information. Functions defined are:

ED *Erase in Display*

Erase data

- From the current position to the end of the display,
- From the start of the display to the current position,
- The entire display.

EL *Erase in Line*

Erase data

- From the current position to the end of the line,
- From the start of the line to the current position,
- The entire line.

CUU *Cursor Up*

Move cursor n lines up. If the cursor reaches the top line, it remains at that position.

CUD *Cursor Down*

Move cursor n lines down. If the cursor reaches the bottom line, it remains at that position.

CUB *Cursor Backward*

Move cursor n columns to the left. If the cursor reaches the leftmost column, it remains at that position.

CUF *Cursor Forward*

Move cursor n columns to the right. If the cursor reaches the rightmost column, it remains at that position.

The following functions are associated with local input editing:

DCF *Delete Character*

Delete character at the current position, shifting trailing text leftwards.

IRM *Insert/Replacement Mode*

Enter and exit insert mode. While in insert mode, data is inserted into the line at the cursor position and the text shifted towards the right, rather than overwriting the previous character in that position.

6.5.1.4 Format Control Group

The format control group determine the character display attributes. Function defined are:

SGR *Set Graphic Rendition*

Initiates one or more of the following graphic rendition options, which remain current until the next occurence of SGR:

- All attributes off
- Bold face or increased intensity
- Underscore
- Blinking
- Negative (reversed) image

6.5.1.5 Field Control Group

The field control group specifies data validation criteria for input fields. The single function defined, specifying a number of attributes, is:

DFS *Data Field Specification*

The data field specification determines the field attributes which hold until the next DFS. Attributes

defined are:

- Protection

  - Protected (read-only)
  - Unprotected (data entry allowed)

- Data Type

  - Alphabetic only (mono or duo case)
  - Alphanumeric (mono or duo case)
  - Numeric only (signed or unsigned)
  - Any character

- Entry Type

  - Mandatory or optional entry
  - Duplication of previous field allowed
  - Default value

- Justification

  - Left or right justified
  - Zero or space filled

6.5.2 DPP Encoding

If the DPP was encoded using control character sequences embedded within the text to represent virtual terminal functions, the destination DPS would be required to re-scan the data to isolate the control sequences, identify the function, and then apply the appropriate transformation. This is essentially a complete repetition of the work performed by the source DPS. However, if both control

functions and data are encoded using field descriptors, advantage can be taken of positional significance to eliminate the necessity to re-scan the data. For this reason, the Item Descriptor (ID) formats, described in Section 5.1, are used to encode the DPP functions.

6.5.2.1 DPP Headers

Data Presentation Services are oriented towards the processing of logical units of information (printer pages, screen images) which are known as User Data Units (UDUs). However, there is no necessity to transfer the UDUs over the network in their entirety as a single PDU; generally it is advantageous to process multiple smaller PDUs to reduce buffer utilisation and allow the overlap of data transfer and end-user processing. Providing that the relationship of each PDU to its corresponding UDU is maintained, an individual terminal may receive each PDU individually, or as a complete UDU, transparently to the source. The DPP header of each PDU contains two 'chaining bits' which specify the relationship of each PDU to its UDU. The header also contains sequence numbers for end-user acknowledgement.

6.5.2.2 DPP Data Formats

The DPP is encoded using the ID formats illustrated in Fig. 6.2. To simplify DPP processing, a different ID page is used for each of the virtual terminal modes. A common encoding of the commands across modes allows a formatting command to invoke different mapping functions for each of the modes by simple changes to the ID decoding tables.

FUNCTION	ID CODE	DESCRIPTION
TXT	I,*n*	*n* characters of ASCII Text
ITS	M,1,*n*	*n* spaces
FNK	M,2,*n*	Function key *n*
SIG	M,3,*n*	Signal *n*
FF	U,1	Fun Feed
CR	U,2	Carriage Return
LF	U,3	Line Feed
NL	U,4	New line
HVP	M,4,*l*,*c*	Horizontal vertical position
TAB	U,5	Tab
HTS	U,6	Horizontal Tab Set
TBC	U,7	Tab Clear
ED	M,5,P_1	Erase Display
EL	M,6,P_1	Erase Line
CUU	U,8	Cursor Up
CUD	U,9	Cursor down
CUB	U,10	Cursor left
CUF	U,11	Cursor right
SGR	M,7,P_2	Set Graphic Rendition
DFS	M,8,P_3	Data Field Specification

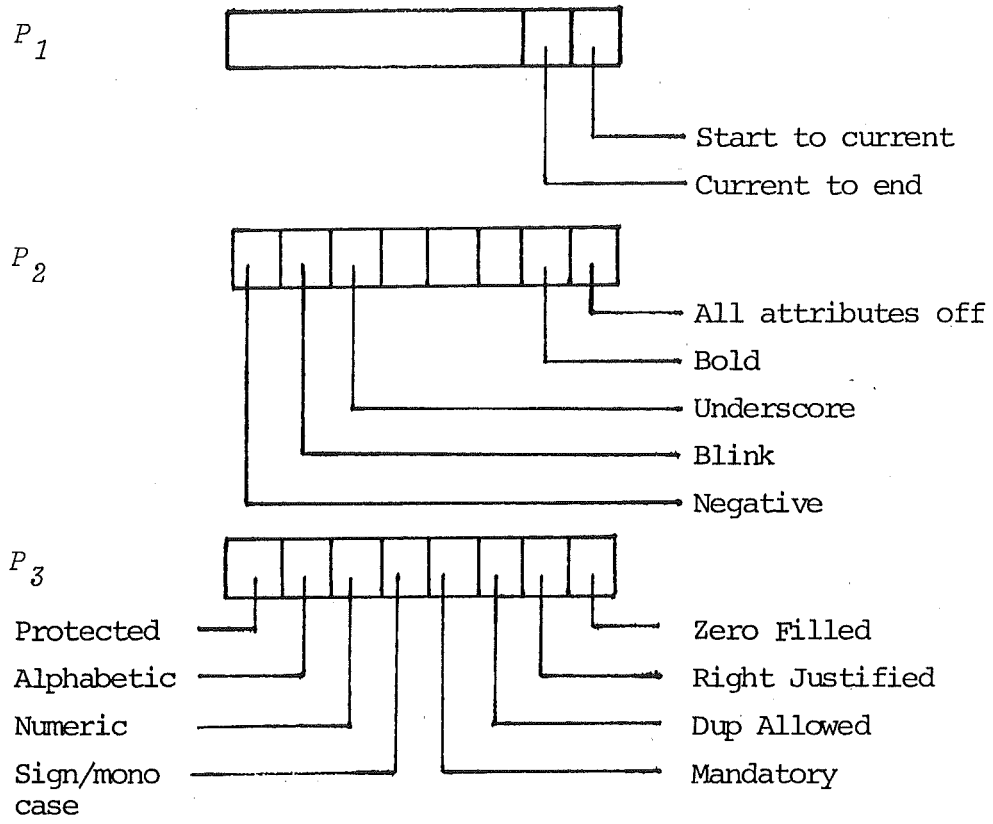


FIG. 6.2 DPP ENCODING

6.6 TERMINAL ATTRIBUTE SPECIFICATION

For each type of terminal device configured, Data Presentation Services require a description of:

- The physical characteristics of the device
- The virtualisation modes which must be supported
- The device control sequences corresponding to the functions of Section 6.5.

This is accomplished by the DCLASS and DEVICE configuration statements. The DCLASS statement is used to define default attributes for classes of devices, while the DEVICE statement auguments these defaults to specify the characteristics of a particular type of device. The configuration statement syntax structure was described in Section 5.5.1. The following discussion of the terminal attribute specification should be read in conjunction with Fig. 6.3.

6.6.1 Universal Device Attributes

Every type of device requires the specification of certain common attributes, specifying the physical characteristics and the basic control functions.

6.6.1.1 Physical Attributes

The physical attributes which must be specified are:

LCLASS — The communications link type associated with the device. This is required for consistency checking of the configuration.

```
ANSI:   DEVICE   PAGESZ=(24,80),TYPE=VIRTUAL,LCLASS=ASYNC,
                 MODES=(SCROLL,DISPLAY),CSI=0'33''[',
                 CURSOR
                 (UP=(CSI,'A')
                 ,DOWN=(CSI,'B')
                 ,LEFT=(CSI,'D')
                 ,RIGHT=(CSI,'C')
                 ,POSN=(CSI,ROW(STRING),';',COL(STRING),'H')
                 ),
                 ERASED
                 (END=(CSI,'0J')
                 ,START=(CSI,'1J')
                 ,ALL=(CSI,'2J')
                 ),
                 ERASEL
                 (END=(CSI,'0K')
                 ,START=(CSI,'1K')
                 ,ALL=(CSI,'2K')
                 ),
                 FORMAT
                 (OFF=CSI,'0m')
                 ,BOLD=(CSI,'1m')
                 ,UNDER=(CSI,'4m')
                 ,BLINK=(CSI,'5m')
                 ,NEG=(CSI,'7m')
                 ),
                 NEWLN=0'15',
                 EDIT
                 (INSERT=0'22'
                 ,DELETE=0'177'
                 ,BACKSP=0'10'
                 ),
                 SIGNALS
                 (EOF=0'32'              % control-Z
                 ,INT=0'03'              % control-C
                 ,SUSPEND=0'023'         % XOFF
                 ,RESUME=0'021'          % XON
                 );
VT52:   DEVICE   DCLASS=ANSI,CSI=0'33',TYPE=CHRVDU,
                 CURSOR
                 (POSN=(CSI,'Y',ROW(ORIGIN=32),COL(ORIGIN=32))),
                 ERASED
                 (END=(CSI,'J')
                 ,START                  % Simulated by software
                 ,ALL=(CSI,'H',CSI,'J')
                 ),
                      .
                      .
                      .
```

FIG. 6.3 TERMINAL ATTRIBUTE SPECIFICATION

TYPE — The device types, which may be:

PRINTER	Asynchronous printing device
CHRVDU	Character-mode (Async) VDU
BLKVDU	Block-mode (Async or Sync) VDU
VIRTUAL	Program interface to the virtual terminal

PAGESZ — The device's page size, specified as (Lines, Columns).

MODES — The allowable modes, which may be SCROLL, DISPLAY, DATA or NATIVE.

Optional attributes are:

XONOFF — The (XON,XOFF) characters for the device, if implemented.

XLATE — Character translation for the device, specified by ('ASCII string', 'target string') couples.

6.6.1.2 Device Control Functions

The printer-oriented page control functions are common to every device. Each is specified by (CHAR, DELAY or SOFT) where:

CHAR — specifies the character code (default is ASCII)

DELAY — specifies the delay period for the function to complete

SOFT specifies that the function is to be emulated by software

The functions are:

FORMFD the 'form feed' character

LINEFD the 'line feed' character

RETURN the 'carriage return' character

NEWLN the 'new line' character, which should only be specified if a single character combines the line feed and carriage return functions

TABCHR the 'horizontal tab' character

6.6.1.3 Process Control Functions

The process control functions require the specification of character sequences which represent:

FNKEYS The set of character strings which represent function keys F1..Fn

SIGNALS The character strings which represent the following events:

EOF	End-of-File on input
INT	Process interruption
SUSPEND	Suspension of output to the device
RESUME	Resumption of output

6.6.2 Device-specific Attributes

The following attribute groups are required for specific types of devices, but not others.

6.6.2.1 Text-Editing Functions

The EDIT group defines the control characters which invoke text-editing functions for character-mode terminals.

DELETE — The 'delete character' code

INSERT — The character which toggles (enters and exits) 'insert mode'

BACKSP — The character which backspaces within the current input

6.6.2.2 Screen-Editing Functions

The ERASE group defines the control sequences for erasing within the display area (ERASED) or the current line (ERASEL). Particular functions are not specified, they are emulated by software. The functions defined are (END, ALL, START) where:

END — Erases from the current position to the end

ALL — Erases all of the line or display

START — Erases from the start to the current position

The CURSOR group defines the cursor positioning sequences for the device. Because there is no consistency in how terminals interpret the UP, DOWN, LEFT and RIGHT functions at the screen

boundaries, explicit positioning sequences must be echoed at these places. The major function is POSN, which specifies the control sequence to move the cursor to a specified coordinate position. The coordinates must be specified with one of two attributes:

ORIGIN — Specifies the origin value corresponding to co-ordinate position 1. If line positions 1 through 24 are specified by the character values 040 through 070 (octal), then the co-ordinate would be specified as ROW (ORIGIN=32).

STRING — The ANSI control sequences require the co-ordinate to be specified by a numeric ASCII string.

6.6.2.3 Format-control Functions

The FORMAT control sequences select the character font of the output device. Different fonts may be inbuilt in the device, or emulated by overprinting (for boldface and underscore). The defined attributes are:

OFF — All attributes off

BOLD — Bold-face

UNDER — Underscore

BLINK — Blinking display (VDUs only)

NEG — Negative (reversed) image (VDUs only)

6.6.2.4 Field-control Functions

The FIELD control sequences determine the attributes of data entry fields within the display. Because there is little

standardisation of data-entry type terminals, only sequences for protected fields are defined:

PROTCT The 'enter protected mode' sequence

UNPROT The 'exit protected mode' sequence

7. EXTERNAL INTERFACES

The external interfaces implement the communications paths between the external devices and the virtualisation functions of the Network Interface Machine

This chapter examines the requirements and design objectives for a generalised communications interface, and the hardware and software structure developed for its implementation. A protocol description language for the implementation of user-specified protocols is described, with examples from the implementation of BISYNC.

7.1 REQUIREMENTS AND OBJECTIVES

Two types of interface are required.

- The character terminal interface assembles messages for the virtualiser, implementing the editing functions described in Section 6.3.1.

- The communications protocol interface implements message oriented communication with external devices using a variety of line types and protocols, such as polled terminals, host computers, or external networks.

This chapter is primarily concerned with the implementation of the latter, although implementation considerations of the character terminal interface are considered.

7.1.1 Functional Requirements

The functional requirements of the external interface are determined by:

- The *attributes* of the physical link,
- The *formats* of the messages passed over the link,

  and
- The *procedures* controlling the interchange of messages over the link.

The major variability is introduced by the functions implemented over the link, which may be:

- Character by character terminal interface
- Character oriented protocol
- Bit-oriented protocol

7.1.1.1 Physical Link Attributes

The physical characteristics of the link are described in terms of the speed (bit rate per second), transmission technique (asynchronous or synchronous) and the character format (data bits, parity and framing bits). While the physical attributes must be described in order to provide the required hardware support functions, this is essentially a selection between a set of predefined options.

7.1.1.2 Message Formats

For a given protocol, all messages may have a common format, with the type being determined by bit fields within the message, or messages may be represented by particular control character sequences, as in BISYNC. Each message type must be distinguished on input, and generated on output, with proper regard for transparency and error detection considerations.

7.1.1.3 Message Procedures

Established procedures for message interchange must be followed to avoid loss or duplication of messages. These include:

- Sending responses or acknowledgements to received messages
- Requesting retransmission when errors are detected during input
- Limiting the allowable time for responses to avoid error lock-up

Procedures must also be followed to control the order of transmission, and select the active station in multi-point configurations.

7.1.2 Structural Objectives

This section develops design objectives for the software components required to implement the external access interfaces.

7.1.2.1 Separation of Framing and Procedures

The framing component which controls the assembly of characters into messages, and the procedural component which directs the flow of messages, should be separate. The interface between these levels should be in terms of the type of message received or to be transmitted, rather than the particular control sequences which represent it. This is required for the following reasons:

- Framing Variations

  Many protocols allow variations in the frame structure to accomodate different character sets and error detection mechanisms. BISYNC allows the use of 6, 7 or 8 bit character codes, and combinations of vertical and longitudinal parity, or cyclic redundancy codes, for error detection. Some X.25 Public Data Networks (PDNs) allow the use of transparent BISYNC envelopes, rather than the HDLC frame envelope, to encapsulate the level 2 frames which enables existing character oriented hardware to access the network.

- Procedural Variations

  Procedural variations can occur due to configuration differences.

  - Operation may be point-to-point (between two stations) over switched or permanent circuits, or multipoint (between a primary and a number of secondary stations).

  - The link itself may be restricted to half-duplex procedures, or full-duplex operation may be allowed.

- some links may have a restricted set of capabilities, while others may use optional facilities

In each case common frame formats may be used; only the procedures controlling the interchange of frames change.

- Functional Isolation

The isolation of the interrupt driven character processing from the more time-consuming but less time-critical message processing should allow an increase in efficiency, and allow the use of specialised hardware assistance for the character processing level.

7.1.2.2 Standardised Framing Interface

In conjunction with the separation of the procedural and framing levels, a standard interface should be provided to permit the interchange of compatible framing and procedural components. Similarly, a common interface should be defined to access the various types of line interface hardware, isolating the framing level from the particular data link characteristics as far as possible. This should be achievable, since single integrated circuits are available which perform asynchronous and synchronous character functions in addition to synchronous bit functions.

7.1.2.3 Hardware/Software Interchange

The software structure should allow the migration of functions between software and hardware for increased performance, particularly at the framing level where functions may be

performed for every character processed. Migration may be performed in three stages:

- Specific Functions

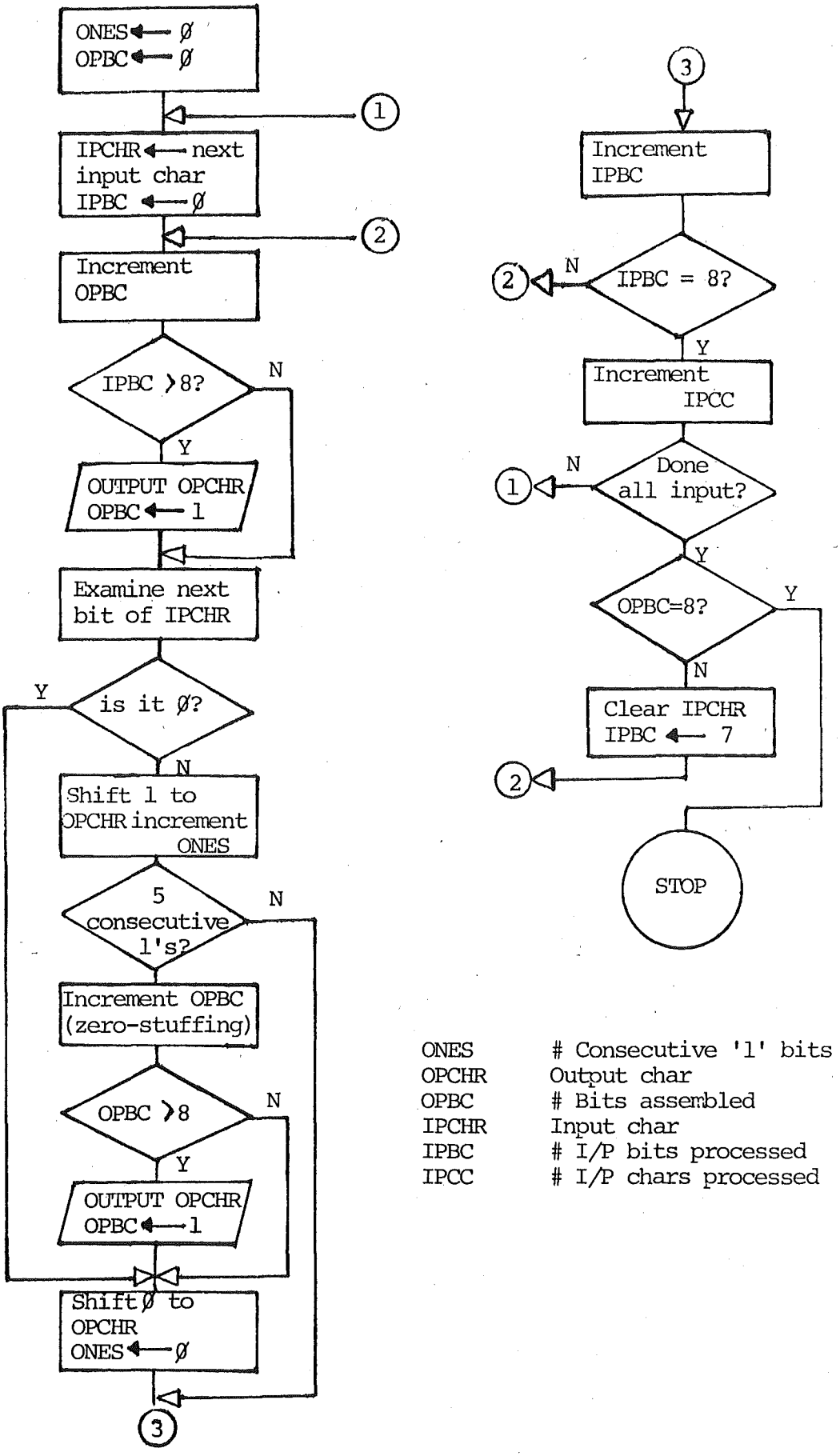
  Some functions, particularly those performed for each bit, are inherently inefficient unless implemented by hardware.

  - Character assembly and disassembly is almost universally performed by special integrated circuits.

  - Cyclic Redundancy Code (CRC) calculation is often implemented by hardware, although reasonably efficient character-oriented table look-up algorithms have been developed.

  - Zero bit insertion/deletion (bit-stuffing) is inherently bit-oriented, requiring complex shifting and masking operations for every character processed.

  An implementation of the bit-stuffing algorithm of Fig. 7.1-A for the 8080 microprocessor is given in Fig. 7.1-B. Although highly optimised, this takes 565 machine cycles per character (190 microseconds on a 3MHz processor) ignoring the additional processing required to fetch and output each character. Some form of hardware assistance is obviously required to handle any appreciable work-load.



ONES	# Consecutive '1' bits
OPCHR	Output char
OPBC	# Bits assembled
IPCHR	Input char
IPBC	# I/P bits processed
IPCC	# I/P chars processed

FIG. 7.1-A BIT-STUFFING ALGORITHM

```
ADDR    CYC     CODE
                        ;+
                        ; BSTUFF - BIT STUFFING ROUTINE
                        ;
                        ; REGISTER USAGE:-
                        ; A                     TEMPORARY (SHIFTING)
                        ; B          OPBC       OUTPUT CHARACTER BIT COUNT
                        ; C          ONES       NUMBER CONSECUTIVE ONE BITS
                        ; D          IPCC       INPUT CHARACTER COUNT
                        ; E          OPCHR      OUTPUT CHAR BEING ASSEMBLED
                        ; H          IPCHR      INPUT CHAR BEING PROCESSED
                        ; L          IPBC       INPUT CHAR BIT COUNT
                        ;-
0000                    BSTUFF:
0000            010508              LXI     B,0805H         ; OPBC=8 , ONES=5
0003            C32900              JMP     BS06            ; GO GET INITIAL CHAR
0006                    BS00:
                        ; FETCH NEXT I/P CHAR INTO H
0006    7       2E08                MVI     L,8             ; # I/P BITS TO PROCESS
0008                    BS01:
0008    4       05                  DCR     B               ; FILLED O/P CHAR YET?
0009    11      C20E00              JNZ     BS02            ; NO - CONTINUE
                        ; OUTPUT ASSEMBLED CHARACTER
000C    7       0608                MVI     B,8             ; RESET # O/P BITS
000E                    BS02:
000E    4       7C                  MOV     A,H             ; ISOLATE
000F    4       0F                  RRC                     ;  NEXT BIT
0010    4       67                  MOV     H,A             ;   OF I/P
0011    4       7B                  MOV     A,E             : GET O/P CHAR
0012    11      D22100              JNC     BS04            ; WAS I/P BIT A '1'?
0015    4       1F                  RAR                     ; YES - SHIFT TO O/P
0016    4       0D                  DCR     C               ; SEEN 5 '1'S YET?
0017    11      C22400              JNZ     BS05            ; NO - DONT STUFF
001A    4       05                  DCR     B               ; ROOM FOR ANOTHER O/P BIT?
001B    11      C22000              JNZ     BS03            ; YES - SKIP
                        ; OUTPUT ASSEMBLED CHARACTER
001E    7       0607                MVI     B,7             ; RESET O/P BIT COUNT
0020                    BS03:
0020    4       B7                  ORA     A               ; CLEAR THE CARRY
0021                    BS04:
0021    7       0E05                MVI     C,5             ; RESET '1'S COUNT
0023    4       1F                  RAR                     ;  & SHIFT 0 TO O/P
0024                    BS05:
0024    4       5F                  MOV     E,A             ; SAVE O/P CHAR
0025    4       2D                  DCR     L               ; ANY BITS OF IPCHR LEFT?
0026    11      C20800              JNZ     BS01            ; YES - GO FINISH IT OFF
0029                    BS06:
0029    4       15                  DCR     D               ; ANY MORE CHARS TO GO?
002A    11      F20600              JP      BS00            ; YES - GO PROCESS THEM
002D    4       65                  MOV     H,L             ; CLEAR I/P CHAR
002E    7       2E01                MVI     L,1             ;  & SET I/P BIT CNT
0030    4       78                  MOV     A,B             ; PARTIALLY ASSEMBLED
0031    7       FE08                CPI     8               ;  OUTPUT CHARACTER ?
0033    11      C20800              JNZ     BS01            ; YES - PAD OUT WITH '0'S
```

FIG. 7.1-B BITSTUFFING ON THE 8080 MICROPROCESSOR

- Intelligent Hardware

  Rather than interrupting the processor to service each incoming or outgoing character, the line interface hardware may be provided with an independent path to memory over which characters can be transferred without processor intervention. To obtain the full benefit of this approach, the line interface hardware must contain some of the intelligence normally provided in the interrupt service routine. Intelligent hardware may be dedicated to a particular protocol (typically the SDLC family), or may be parameterised for a range of protocols.

- Distributed Processing

  A further approach is to distribute the functions between processors, so that the processor-intensive character functions are performed by smaller 'slave' processors which interface to a master processor performing the higher level control functions. The slave processors can be programmed to perform a wide variety of functions, isolating the master processor both from the workload and the internal details of the functions.

7.1.2.4 Standardised Queue Structures

Each station on a link requires queues to buffer input and output messages. The current station may be selected either from the station address returned in a poll response, or by station attributes such as 'next station with output queued'. Stations may be selected cyclically, or in some priority sequence.

Each queue may require sequence numbers to be associated with the messages (such as the X.25 send and receive numbers). It may also be necessary to enqueue high-priority messages at the head of the queue rather than the tail, and to retain messages on the queue until an acknowledgement has been received, rather than removing them as they are fetched. A queue may also require an associated timer to detect inactivity and possible error situations.

7.2 LINE MODULES

A *line module* is a hardware and software component which implements the complete framing functions for a protocol. It is characterised by:

- Providing a uniform interface to the procedural level independently of the particular framing functions performed
- Performing all character handling functions required to assemble complete input messages, and transmit messages
- Being designed to exploit the capabilities of line hardware from the simple character interfaces to programmable slave processors
- Protocol independence through the use of control tables to define the framing functions

This section describes the design of a universal line module, covering

- The line interface hardware characteristics

- The design of decision tables to control message assembly

- The details of message input and output framing functions

- The interface to the procedural level

7.2.1 Line Interface Hardware

A variety of hardware options are necessary to allow the most optimal implementation of a particular communications interface. This section describes the characteristics of:

- Asynchronous and synchronous interfaces

- Dedicated 'protocol chips'

- Programmable slave processors

7.2.1.1 Asynchronous Character Interfaces

The *Universal Asynchronous Receiver/Transmitter* (UART) is a single LSI circuit which performs the conversion between characters and asynchronous formatted bit patterns on a line. The programming interface provides:

- A *receiver buffer register* for the assembled input character

- A *receiver status register*

- A *transmitter buffer register* containing next output character

- A *transmitter status register*

- *Parameter register(s)* specifying the line attributes

7.2.1.2 Synchronous Character Interface

The *Universal Synchronous Receiver/Transmitter* (USRT) is similar to the UART, except that the conversion is between characters and synchronous bit patterns. Four special synchronous line control facilities are provided:

- A *sync search bit* which forces the receiver logic to re-establish synchronism by searching for at least two consecutive SYNC characters

- A *sync strip bit* which causes the receiver to discard SYNC characters between the initial SYNC sequence and the first non-SYNC character

- An *underrun flag* set when the next output character was not loaded by the time the transmitter required it

- A *sync register* which contains the SYNC character value

Some synchronous interfaces may also implement CRC calculation and checking, and 'transparent mode' facilities for BISYNC-like protocols. Both synchronous and asynchronous transmission functions may be combined in a USART.

7.2.1.3 Synchronous Bit (SDLC) Interfaces

The *Data Link Control* (DLC) circuits, typified by the Intel 8273, perform the transmission and reception of bit-oriented

protocol frames (Section 3.1.2). The chips implement:

- Message framing
- CRC generation and checking
- Zero bit insertion and deletion
- Flag, Idle and Abort processing

Most DLC chips provide a 'secondary address' register which may be used to discard all frames other than those addressed to the specific station.

7.2.1.4 Programmable Slave Processors

The *Universal Peripheral Interface* (UPI) is a complete micro-computer which functions as a programmable 'slave' controller to a host processor. The Intel 8741 UPI contains:

- An 8-bit CPU with over 90 instructions

  Most instructions are single-byte and execute in 2.5 microseconds. The instruction set is optimised for bit manipulation, input-output, table look-up and n-way branches. Timer and input-buffer full interrupts are prioritised, and the two register sets facilitate interrupt processing.

- 1024 bytes of read-only memory

  The processor contains a special instruction to access tabular information in the uppermost 256 bytes of memory.

- A programmable interval timer

- Two 8-bit bidirectional ports

  Each bit is individually controllable for input and output.

- A host-processor interface

  Buffer and status registers are provided to communicate with the host processor.

The UPI can also be provided with a Direct Memory Access (DMA) path to memory for host-independent data transfer.

7.2.1.5 Modem Control

A *modem* is a device which converts between the binary bit patterns of the line interface hardware and a form more suitable for long distance communications. The modem is coupled to the line interface hardware by control and status leads, as well as the data lead. A modem control register is required in the interface to provide (for non-switched lines) the following status signals:

DSR *Data Set Ready*

DSR indicates that the modem is powered on and is ready to transmit data.

CD *Carrier Detect*

CD indicates that a signal is being received from the distant modem.

RTS *Request to Send*

RTS initiates a transition from receive to transmit modes, commonly known as line turn-round.

CTS *Clear to Send*

CTS is an indication from the modem that the turn-round has completed. It is generated a predetermined time after the RTS flag has been asserted by the computer.

7.2.2 Input Processing Functions

On input, the line module distinguishes control sequences and delimits messages within the input character stream. For each character, a predetermined set of actions must be performed.

7.2.2.1 Table-Driven Framing

Each protocol has special rules about which characters

- Start or terminate messages
- Are significant as data
- Are included in the error detection calculation
- Affect the interpretation of following characters
- Require special handling

There are two possible ways of controlling this processing.

- *Decision sequences* determine, for each possible *action*, whether or not that action is to be performed for the particular input character.

- *Decision tables* determine, for every *character*, the set of actions which are to be performed. The incoming character stream may be regarded as a sequence of macro-instructions (characters) selecting micro-instructions (table entries) which are interpreted to perform the required actions.

Interpretation usually has the implication of increased overhead. However, careful table design may result in reduced overhead compared with explicit decision sequences. If the table entry for a character is masked by the allowable actions for the current state, it is possible to determine in a single operation whether the default processing is to be performed, or special actions are required. The specific actions can be determined by examining successive bits in the mask, rather than by repeated range tests on the character value for each action. It is possible that, for a particular protocol, not every action required can be specified within the table entry. However, a 'special attention' bit can be provided to invoke special processing routines for the exception situations.

The use of decision tables also separates the actions themselves from the control of the actions for any particular protocol. This allows general line modules to be developed which can be loaded with tables for specific protocols.

7.2.2.2 Required States and Actions

Each decision table entry must specify which of the allowable actions are to be performed for the character. This selection may be modified by the current mode (state) which is a function of previous control characters. Each table entry must specify whether the character:

- May start or end a message
- Is stored in the message or discarded
- Is included in the Error Check Calculation (ECC), or excluded
- Initiates or terminates the ECC
- Requires special processing
- Initiates a mode change

The three basic modes required to meet the requirements of most common protocols are:

- *Normal mode*, in which the standard attributes determine the processing of the character
- *Shift mode*, entered on detection of a special shift character, in which an alternative set of attributes is used to determine the processing of the following characters

  and

- *Transparent mode*, in which following characters are

processed completely transparently, being stored and accumulated into the ECC, unless preceded by the 'shift' character. In this case the character is processed according to the normal-mode attributes.

The corresponding state transitions are illustrated in Fig. 7.2.

In addition to the individual character table entries, an overall control mask may be used to disable or modify some of the actions; for example, whether the ECC includes the initiating character, or commences with the following character.

7.2.3 Input Protocol Tables

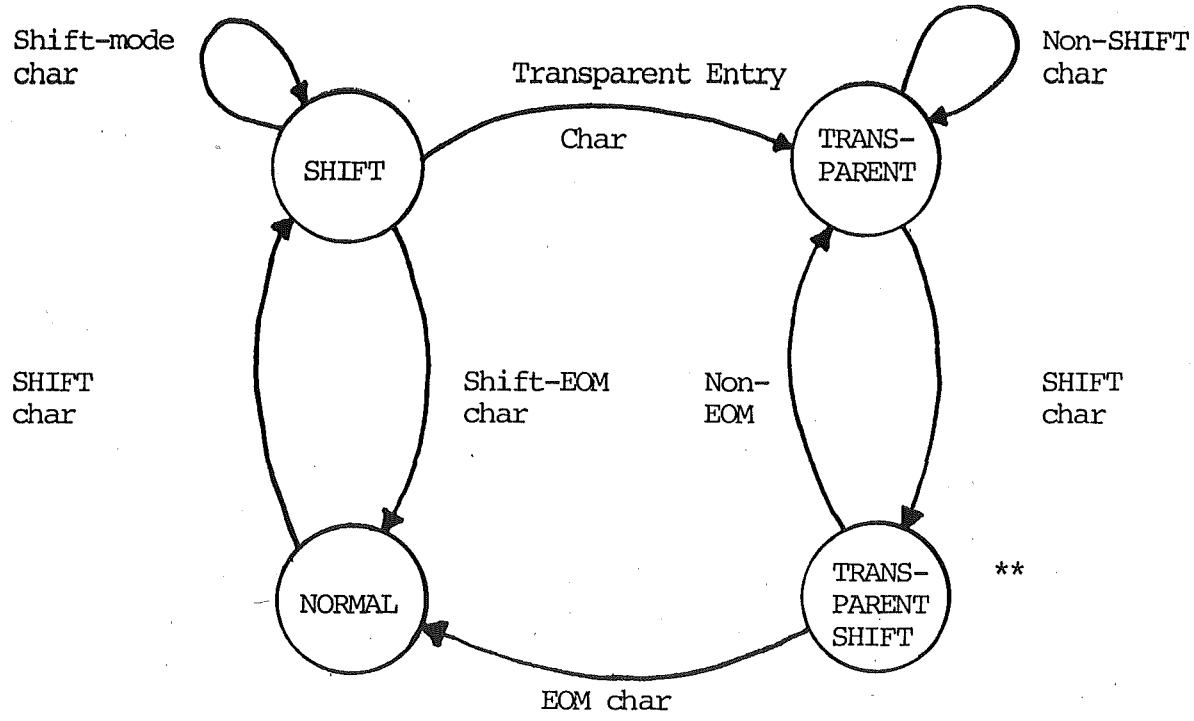
To specify the framing for a protocol, three sets of information must be provided. These are:

- The Character Control Table (CCT), which contains the 'micro instruction' entry specifying the processing for each character

- The Control Modified Mask (CMM), which enables or modifies the processing of actions specified in the CCT entries

and

- The Shift-In Character (SIC), which is used to enter the shift or transparent-shift states.



** Temporary state

FIG. 7.2 STATE TRANSITIONS

7.2.3.1 Character Control Table Entry

Each CCT entry (Fig. 7.3) consists of 8 bits controlling the processing for the particular character index into the table. The interpretation of these bits is described below.

- The *end of message (EOM)* bit indicates that the character terminates the message. Status information is stored in the message control block, the buffers linked into a queue of completed inputs, and a new buffer initialised for subsequent input data.

- The *discard* bit indicates that the character should be discarded, rather than being stored into the message buffer.

- The *exclude* bit indicates that the character should be excluded from the ECC, rather than being accumulated.

- The *start of text (SOT)* bit indicates the start of message text. The text pointer is initialised to the next character position of the buffer, which will be occupied by the first text character.

- The *start check* bit indicates that the ECC should be initiated if not already active. The inclusion of the initiating character is determined by the CMM *initial include* bit.

- The *receive check* bit indicates that the longitudinal Block Check Character (BCC) or Cyclic Redundancy Code (CRC) follows this character. It is usually set in conjunction with the *EOM* bit.

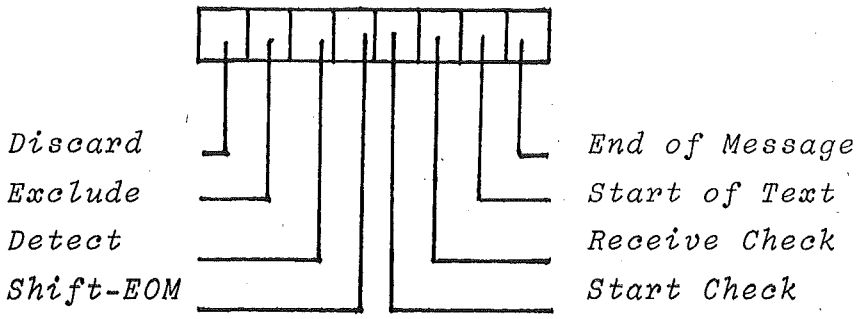


FIG. 7.3 CCT ENTRY

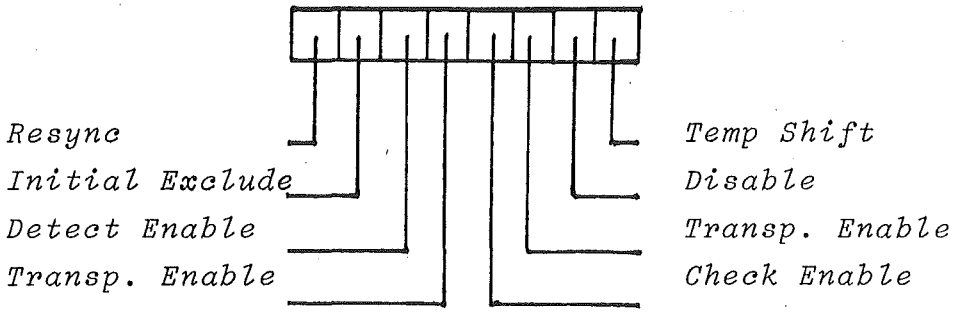


FIG. 7.4 CMM FLAGS

- The *detect* bit indicates that the character requires special processing.

- The *shift-EOM* bit is interpreted only when a preceding 'shift' character has been detected. When set, it indicates the end of the message. When clear, its interpretation depends on the CMM setting. If transparent mode is enabled, it indicates entry into transparent mode; otherwise it indicates continued shift-mode processing.

7.2.3.2 Control Modifier Mask

The Control Modifier Mask (CMM) (Fig. 7.4) enables or modifies the processing of particular functions within the CCT entries, and provides global control functions. The CMM bits are:

- The *resync* bit indicates that any input accumulated since the preceding end-of-message should be discarded. For synchronous lines, a 'synch search' is initiated.

- The *disable* bit indicates that all input should be ignored while it remains set.

- The *temp shift* bit controls the operation of shift mode. When set, it indicates that the shift should apply to a single character only, returning to normal mode if the character following the 'shift' does not have *shift-EOM* set. When clear, shift mode is maintained until a character with the *shift-EOM* bit set is detected.

- The *transparent include* bit indicates that the 'shift' character should be accumulated into the ECC while in transparent mode.

- The *initial exclude* bit indicates that the character initiating the ECC should be excluded from the calculation; otherwise the initial character is included.

- The *check enable* bit indicates that the longitudinal ECC (BCC or CRC) is enabled; otherwise the *start check* and *receive check* bits in the CCT entries are ignored.

- The *detect enable* bit indicates that the special character detection is enabled; when the *detect* bit is encountered in a CCT entry special processing is initiated.

- The *transparent enable* bit controls the interpretation of the *shift-EOM* bit in the CCT entries.

  When it is set, a clear *shift-EOM* bit indicates a transition into transparent mode. While in transparent mode, any vertical parity checking is disabled.

  When it is clear, a clear *shift-EOM* bit indicates a continuation of shift mode or a return to normal mode, depending on the *temp shift* modifier.

  Regardless of transparent enable, a set *shift-EOM* indicates the end of the current message.

7.2.3.3 Shift-In Character

The Shift-In Character (SIC) is used to specify the character code which initiates the transition from normal to shift mode, or from transparent to transparent-shift mode. While in transparent mode, all characters are regarded as data, being stored and accumulated into the ECC, unless preceded by the Shift-In Character when the CCT entry defines the operations to be performed.

7.2.4 Input Character Handling

Interrupt processing functions may be performed for every character entering and leaving the system. The efficiency with which interrupts are processed will have a significant influence on the overall performance of the NIM.

7.2.4.1 Performance Objectives

Design goals for the interrupt processing code are:

- Minimise the interrupt service latency time

  The software response time will be largely determined by the maximum period during which interrupts are disabled, which must be minimised. This may be achieved by separating interrupt processing into two stages; the initial processing under interrupt lockout, and the remainder with interrupts enabled.

- Minimise the service duration of interrupts

  The time to service an interrupt will be the sum of the interrupt service routine entry and exit times,

and the time to process the character. The time to save and restore contexts is essentially non-productive overhead, but may take a similar period to the actual character processing. Where possible, context saving must be avoided.

- Spreading workload over time

  The time period over which an operation can be deferred must be maximised, to allow the smoothing of workload peaks. This may be achieved by buffering (queueing) between processing steps. If possible, the buffering should not increase the service time significantly.

- Reduced overheads at high activity levels

  Where possible, processing overheads should increase at a slower rate than the workload. If several units of work may be performed by a single activation of a function at high activity levels, the entry and exit overhead is shared between the units. This may be achieved by queueing work to a function.

7.2.4.2 Interrupt Processing

Many processors provide two hardware contexts; one for interrupt and the other for background processing. Both the Z-80 and the 8741 UPI processors have this feature. When this is not provided (as in the 8080) overhead can be minimised by only using a subset of the registers during interrupt processing, as long as inefficiencies are not incurred by the reduced number of usable registers.

The interrupt processing flow is illustrated in Fig. 7.5. The algorithm satisfies the design objectives, and functions

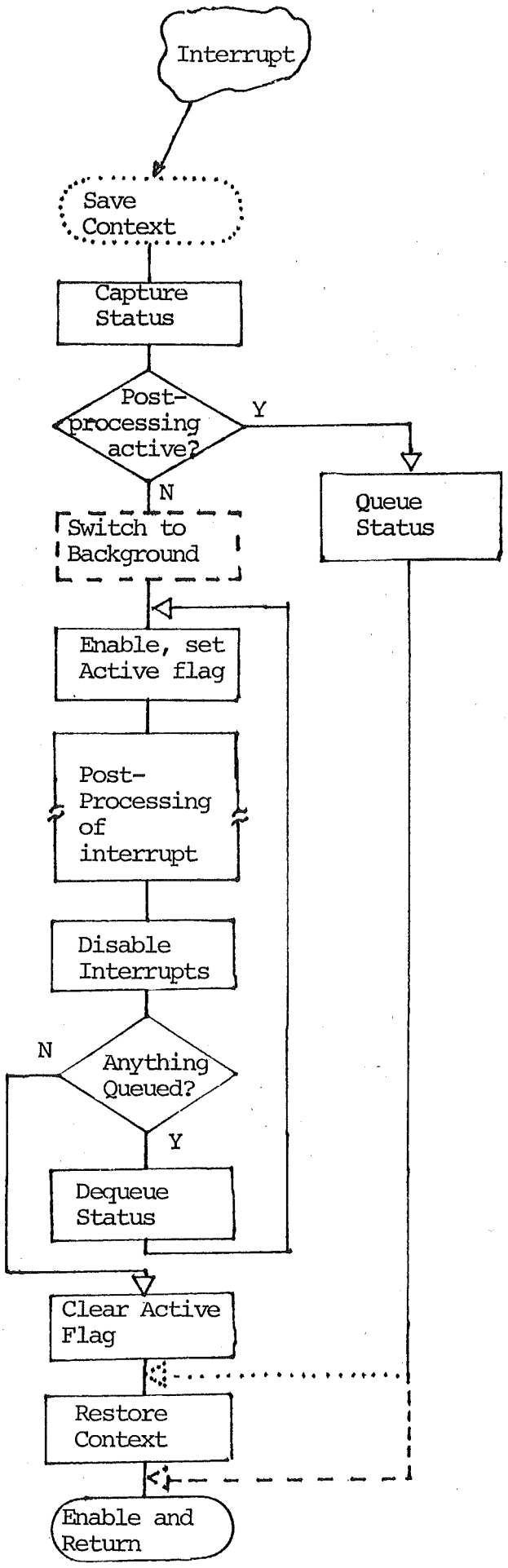


FIG. 7.5 INTERRUPT PROCESSING ALGORITHM

efficiently in both single and dual context environments. Interrupt processing is divided into two phases.

- *Pre-processing* captures the hardware status under interrupt lockout.

- *Post-processing*, which performs the majority of the work interpreting the CCT tables, executes with interrupts enabled. At low activity levels direct paths avoid queueing overhead, while queueing at higher activity levels reduces the overhead of saving and restoring contexts significantly.

7.2.5 Output Processing

Output processing is significantly simpler than input processing. Where input processing must be able to recognise any of the possible message types at any time, and may not know the type until the input has completed, on output the message type and characteristics are known from the outset.

7.2.5.1 Functional Requirements

An output message generally consists of a sequence of control characters, or data enveloped by control sequences at the beginning and end. The ECC may need to be performed over sections of the message, and transmitted at some point (normally the end). Transparency functions may need to be applied over sections of the message. Copying of messages to add control sequences should be avoided, and attributes should be applied to message segments, rather than on a character basis, to allow the use of Direct Memory Access (DMA) devices.

7.2.5.2 Buffer Descriptors

The above requirements are satisfied by the use of buffer descriptors to define the output processing. Each buffer descriptor defines a segment of a message in terms of its memory address, length, and attributes. A sequence of buffer descriptors with different attributes may be cascaded to frame a complete message.

The buffer descriptor format is illustrated in Fig. 7.6. The control bits defined are:

- The *end-of-message* bit indicates the last buffer descriptor in the chain. If the bit is clear, another buffer descriptor follows.

- The *start check* bit indicates that the ECC should be initiated with this segment.

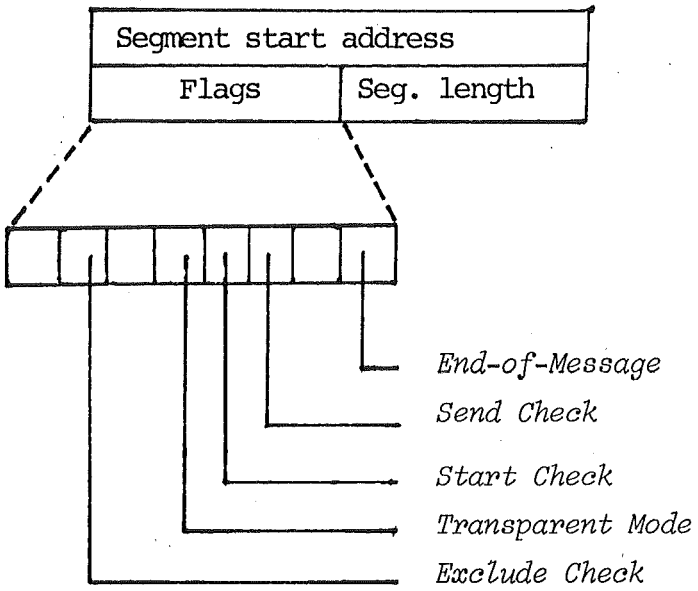
- The *exclude check* bit determines whether the data in the segment is accumulated into the ECC.

- The *send check* bit indicates that the accumulated ECC should be transmitted following this segement.

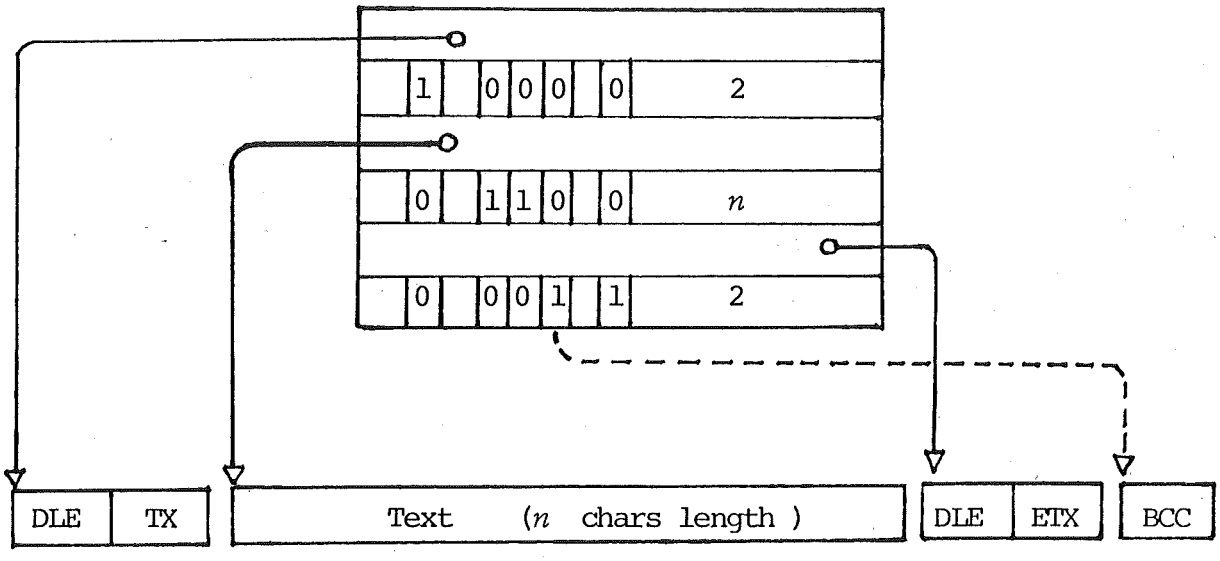
- The *transparent mode* bit indicates that the segment is to be sent in transparent mode. Any 'shift' character will be preceded by another 'shift'.

7.2.5.3 Output Character Handling

The objectives for output interrupt processing are similar to those of input processing. However, since output operations require less processing than input operations,



BCW FORMAT



Example: BISYNC TRANSPARENT FRAME

FIG. 7.6 BUFFER CONTROL WORDS

maximum performances may be obtained using simpler techniques. In many situations character by character interrupts may be avoided by providing the interface with a direct path to memory. Where necessary, the output message can be pre-scanned and broken into segments which require no special processing. Where character by character processing must be performed, it is necessary to minimise the interrupt service latency time (especially for synchronous lines which must transmit characters at fixed time intervals), and the interrupt lockout and total processing duration. This is accomplished by separating the interrupt routine into pre-processing and post-processing functions. The next character to be transmitted is always 'pre-fetched' and held in a temporary storage location. When an output character request interrupt occurs, this character is loaded immediately into the hardware register, and then the next character pre-fetched.

7.2.6 Line Module Interface

A major function of the line module is to provide a uniform interface to the procedural level regardless of the particular hardware or framing functions performed. Previous sections described how the protocol framing functions may be specified using decision tables for input, and buffer descriptors for output. This section describes the interface formats to the procedural level.

7.2.6.1 Interface Functions

The line module interface performs two functions. It is used to condition the line module for the particular line attributes, and to initiate the data transfer operations.

The procedural level initiates line module functions by calling the line module handler routine, passing the address of a command table specifying the function to be performed. The line module handler examines the function, then performs the function directly for non-intelligent hardware, or passes the command to the slave processor. On completion, status information is returned in the command table.

7.2.6.2 Input/Output Operations

For input and output operations, the command table is six bytes long. It contains:

- Function

  The first byte specifies the function to be performed. The functions are:

  Input

  Return the next input message

  Resync Input

  Discard accumulated input, resynchronise and return the following input message

  Control Output

  Output the specified control message

  Data Output

  Output the data (text) message

- Aux Function

  The auxiliary function specifies the timeout value on input, or the (protocol dependent) message type on output.

- Status

  The status byte returns standard status conditions at the completion of the operation.

- Aux Status

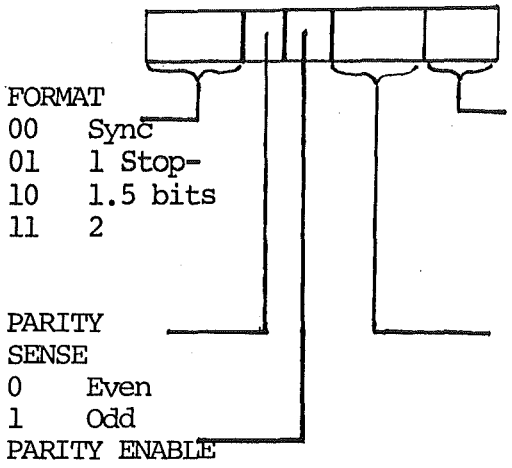
  The auxiliary status either returns extra status corresponding to a status bit (eg. modem status), or the input message type for a good input (all status bits clear).

- Buffer Address

  The buffer address points to the data buffer chain for output. For input it is used both to pass empty buffer chains to the line module on initiation of an input function, and to return the message buffer address on completion of a good input.

7.2.6.3 Control Operations

For control functions, the command table contains a variable number of bytes, determined by the particular function. The function byte specifies the operation to be performed, followed by a function-dependent number of parameters. For the Set Config function used to initialise the line hardware, the parameters specify the character and error check formats, speed, and shift character.



PARAMETER REG

FORMAT
00 Sync
01 1 Stop-
10 1.5 bits
11 2

CHECK
00 No check
01 BCC
10 CRC-16
11 CRC-CCITT

PARITY SENSE
0 Even
1 Odd

PARITY ENABLE

CHAR LENGTH
00 5 bits
01 6
10 7
11 8

Tx Speed	Rx Speed

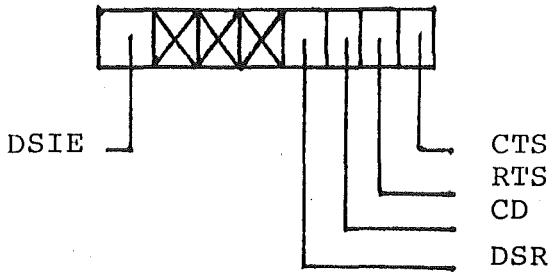
SPEED/SYNC REG

Octal	Speed (BPS)
00	0
01	50
02	75
03	110
04	134.5
05	150
06	200
07	300
10	600
11	1200
12	1800
13	2400
14	4800
15	9600
16	19200

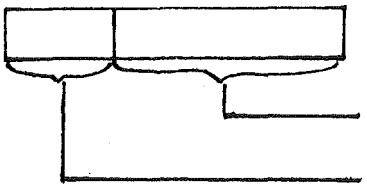
Shift char

SHIFT CHAR REG

FIG. 7.7-A LINE MODULE PARAMETERISATION

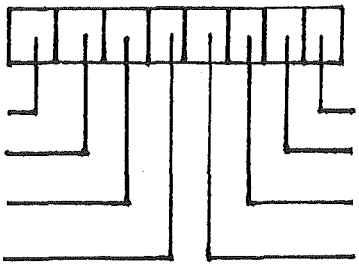


MODEM CONTROL

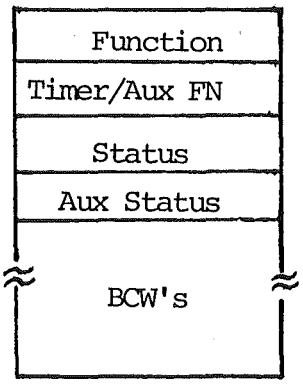


00 milliseconds
01 10 milli
10 100 milli
11 seconds

TIMER



STATUS



COMMAND BUFFER

FIG. 7.7-B LINE MODULE COMMANDS

7.3 LINE PROCESSES

The line processes perform the complete message-level procedural functions for a protocol.

The implementation of the line processes is simplified due to the strict separation of procedural and framing functions. As the message formats are determined by the framing functions implemented within the line modules, the procedural level is concerned solely with:

- Station selection and transmission sequencing
- Sending and receiving data messages
- Sending and processing acknowledgements and responses
- Initiating and processing retransmissions for error recovery
- Timing input requests to avoid lock-up situations

This section describes the internal structure of the line processes, covering:

- The message queueing structures
- The internal process structure
- The interface to Session Control and the Line Modules

7.3.1 Queue Control

Queue management is central to the Line Processes. Queues are required to hold the input and output messages for each station on the link, and to provide temporary storage of transmitted messages while awaiting acknowledgements. Functions are required to select the active station queues according to various attributes, and to insert and detail items from the queue.

7.3.1.1 Queue Structure

A simple and universal queue structure is used throughout the Network Interface Machine. This structure provides temporary storage during acknowledgement and message sequence numbering, as well as being efficient and easily implemented on microprocessors with restricted instruction sets.

The queue structure, illustrated in Fig. 7.8, consists of:

- The Queue Control Block (QCB) contains information to permit manipulation of the queue items.
- The Queue List (QL) contains the data items on the queue. These items may be literal values, or addresses of messages.
- The Queue Vector (QV) combines a set of queues into a group for control purposes.

The QCB is the key to queue control, containing the fields which maintain the state of the queue. These are:

SIZE Specifies the maximum number of items allowed in the queue.

THIS Indicates the current queue item for processing. THIS may range from zero to SIZE, and the least significant bits correspond to the message sequence number.

PREV Indicates the number of items in the queue that have been previously processed, but are still to be acknowledged.

NEXT Indicates the number of items in the queue that are waiting to be processed. NEXT may range from zero to (SIZE - PREV).

ADDR Specifies the starting address of the Queue List. To simplify the queueing functions, queues are restricted to a maximum of 128 items, and a Queue List is contained within a single page of memory (between addresses xx00 and xxFF).

TIMER The timer is used to time-out activities on the queue.

Individual queues are grouped together for control purposes by a Queue Vector (QV). The QV contains the address of the input and output QCB's for each queue in the group, and indexes specifying the currently selected input and output queues. Each individual queue is referenced by its index within the QV, and a queue may be contained within multiple QVs. Typically a queue will be accessed by two QVs, one for inserting and the other for retrieving items.

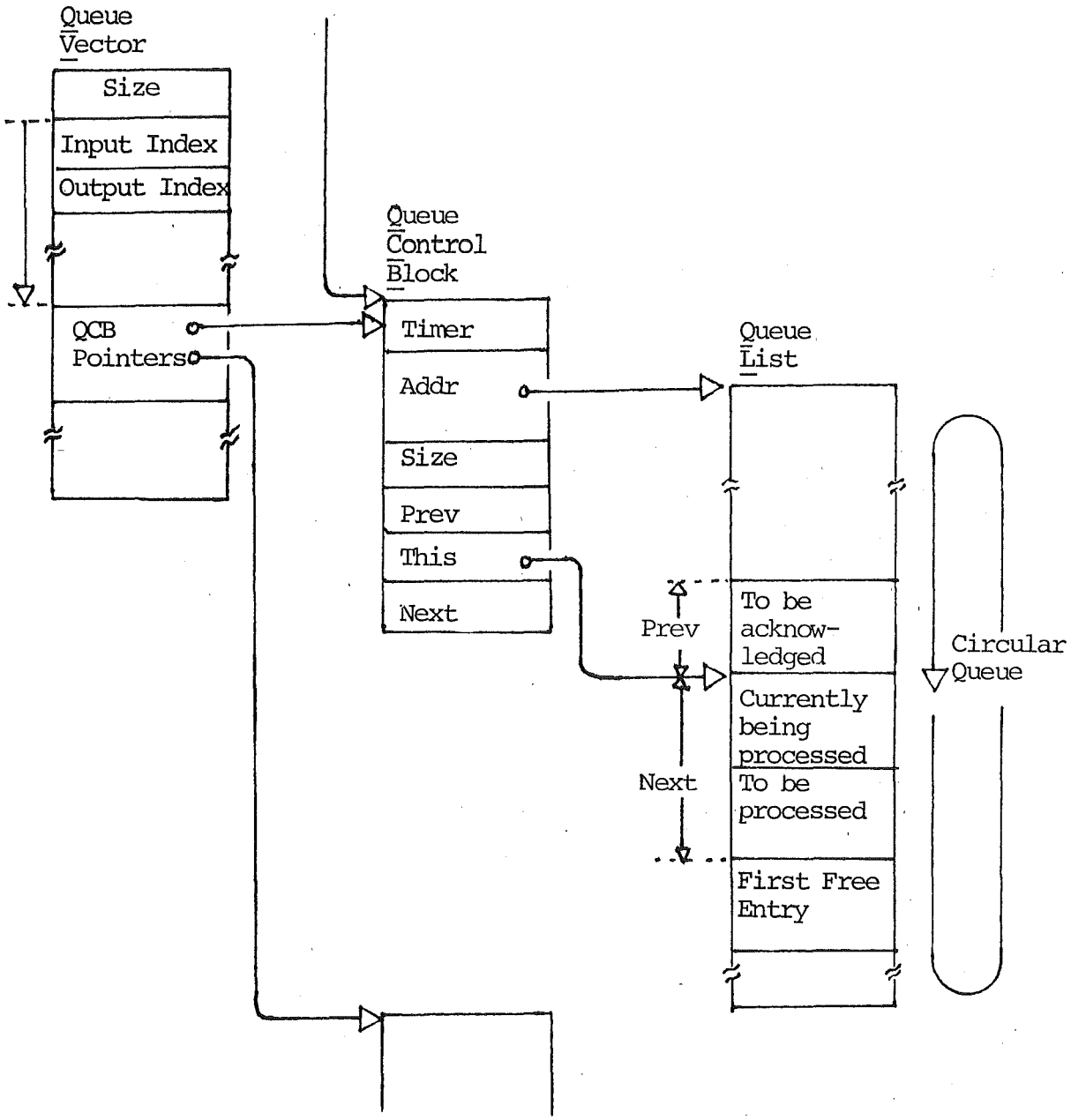


FIG. 7.8 MESSAGE QUEUE STRUCTURE

7.3.1.2 Queue Functions

Items on a queue are normally processed on a sequential first-in/first-out (FIFO) basis. However, it is possible to insert high-priority items at the front of the queue, and to reprocess previously accessed but unacknowledged items. The queue manipulation functions are:

- Add item to the tail of the queue
- Insert item at the front of the queue
- Fetch the front item from the queue
- Acknowledge previously-fetched items
- Requeue previously-fetched items
- Check queues defined by the queue vector for unprocessed items
- Cyclically select queues within the queue vector

When items are being added to a queue, a error condition will be detected when the sequence number is not equal to the next expected message number. Likewise, when items are being acknowledged, an error condition will be detected if the acknowledgement sequence number is not within the range (THIS - PREV) to (THIS - 1).

7.3.2 Process Control

Line processes perform the complete procedural functions for a protocol, controlling the allocation of the link to a station, data transmission and reception, and generation of control messages or responses.

7.3.2.1 Process Allocation

Although half-duplex protocols require only a single control process, full duplex protocols are controlled by logically separate processes for input and output. Because input is buffered, a single process could control a full duplex link, but this would result in increased complexity. The use of separate processes requires some form of synchronisation mechanism for coordination, but this may be achieved quite simply by an adaptation of the co-routine mechanism since at any instant the transmit and receive processes must exclude each other as they access shared data.

7.3.2.2 Internal Structure

Line processes are event-driven. A process may be activated by an external event, such as the arrival of an input or output message, or by an internal event associated with conditions within the processes themselves, such as timer expiration. The line processes is essentially a Finite State Machine (FSM). Each situation in which the process is deactivated represents a state, and each event paired with its corresponding actions represents a transition. Transitions may lead back to the same, or different states. The line processes structure is based on the implementation of this state transition model.

Apart from the state transition control structure, which incorporates message queueing, synchronisation, and input-output, the process functions are much the same as any other systems program. Simple arithmetic and logical expressions and decision statements are required, and the ability to define the memory layouts of packed data structures (message headers) and per-line and per-station working storage.

7.3.3 Input-Output Interfaces

The line processes are effectively a programmed data interface, for the transfer of messages between Session Control and the Line Modules.

7.3.3.1 Line Module Interface

Functions are required to transmit frames to the line module, and receive input frames from the line module. A frame may consist of control information only, or may contain user data. The frame type must be specified on output, and returned on input along with status information. For input operations, a maximum delay time must be specified. The line module interface has been previously described in Section 7.2.6.

7.3.3.2 Session Control Interface

Functions are required to fetch output messages from, and to queue messages to, Session Control. In addition, control information must also be passed between Session Control and the line processes to implement session establishment and termination functions, control the flow of data between the access interface and Session Control, and indicate unrecoverable communications errors detected by the access interface.

7.4 PROTOCOL SPECIFICATION

The previous sections have described the internal structure of the LineModules and Protocol Processes. This section describes the specification language constructs which define the characteristics of the external access interface. While

the other layers of the NIM (Session Control and Data Presentation) have well-defined functions which only require predefined attributes to be specified, the external interfaces require a more flexible mechanism since not only must the physical characteristics of the interface be specified, but also the logical processing functions. The three areas which each require a different type of specification are:

- The physical line attributes
- The message formats
- The protocol procedures

7.4.1 Physical Line Attributes

The LCLASS and LINE configuration statements are used to specify the characteristics of the communications links. These statements use the configuration statement syntax previously described in Section 5.1.

7.4.1.1 LCLASS Statement

The LCLASS statement is used to define the default attributes for classes of communications lines, which include:

TYPE	The basic physical characteristics of the Link	
	ASYNC	Asynchronous
	SYNC	Synchronous
	BIT	Bit-oriented synchronous
	SIMPLEX	Alternate transmission and reception
	DUPLEX	Simultaneous transmission and reception

CHARSZ The width (in bits) of the data character, disregarding parity. This may be 7 or 8.

PARITY The Parity checking to be applied on a character-by-character basis, which may be NONE, EVEN, ODD, MARK or SPACE. If other than NONE is specified, an additional bit of the correct sense is appended to each character, transmitted and checked on each character received.

SYNC Specifies sync-character processing for synchronous lines. The attributes are:

CHAR	The sync-character code
STRIP	Whether characters are to be discarded on input

SPEED The line speed. (Bit rate per second).

MODEM The modem type, which must be defined by a compatible MODEM statement, or defined as DIRECT if no modem is used.

PROTOCOL The default protocol for the line.

ADAPTOR The line interface hardware type.

7.4.1.2 LINE Statement

Each LINE statement specifies a specific instance of a communications link. General attributes may be supplied from a default LCLASS, or may be explicitly defined. In addition, the following attributes specific to each particular line must be defined:

ADDR The address used to select the line interface hardware.

PROC The processor with which the line interface hardware is associated.

7.4.2 Message Format Specifications

Language constructs are required to specify message formats, which are used to build the Character Control Table (CCT) for input, and the Buffer Control Word (BCW) chains for output.

7.4.2.1 FRAMES statement

The FRAMES statement is used to define global parameters for a protocol. The attributes which may be defined are:

SHIFT The shift-character for transparent mode.

CRC The Cyclic Redundancy Code type; attributes are

- POLY The CRC polynomial (CRC16 or CCITT).
- INIT The initial CRC value (ONES or ZERO).
- INVERT Whether the final value is inverted.
- INCLUDE Whether the initiating character is included in the CRC.

BCC The Block Check Character type; attributes are

- SENSE The longitudinal parity sense (ODD or EVEN).
- PARITY The parity of the BCC character itself. LONG specifies that the BCC parity bit

is the sum of the parity bits of the message characters, while VERT indicates that the BCC parity bit is a true parity bit.

INCLUDE — Whether the initiating character is included in the BCC.

MAXMSG — The maximum allowable message length.

7.4.2.2 MSG statement

The MSG statement is used to define the format of each type of message as a sequence of elements with their associated attributes, as illustrated in Fig. 7.9. The three types of elements are:

char — An individual character code, normally a control character.

HEADER — A sequence of characters which are stored in the message control block until a start-of-text or end-of-message character is received.

TEXT — The user data, which is stored in the message buffers until an end-of-message character is received.

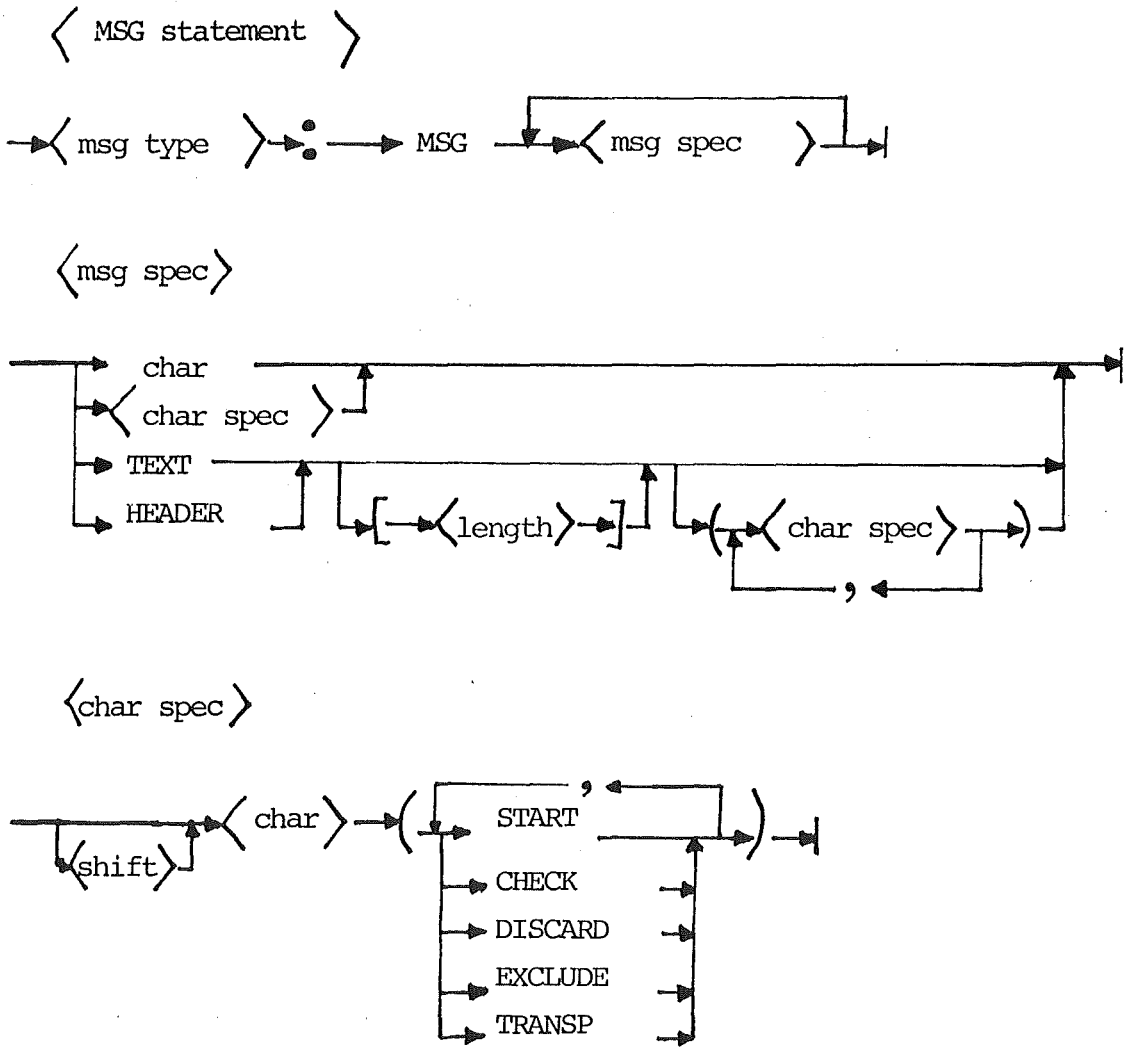
Both HEADER and TEXT may have special actions specified for particular characters occuring within the data, as with *char*. The actions, which are used to encode the CCT entries, are:

START — Start BCC/CRC accumulation

CHECK — Receive or send BCC/CRC

DISCARD — Discard the text character without storing

SYNTAX:



EXAMPLES:

```
BSC:          FRAMES   SHIFT=DLE,MAXMSG=256,
                       CRC(POLY=CRC16,INIT=ZERO,INCLUDE=FALSE);
BSCEOT:       MSG      EOT;
BSCACKØ:      MSG      DLE 'Ø';
BSCACK1:      MSG      DLE '1';
BSCNAK:       MSG      NAK;
BSCTXT:       MSG      STX(START)
                       TEXT(SYN(DISCARD,EXCLUDE))
                       ETX(CHECK);
BSCXTXT:      MSG      DLE STX(START,TRANSP)
                       TEXT(DLE SYN(DISCARD,EXCLUDE))
                       DLE ETX(CHECK);
```

FIG 7.9 MESSAGE FORMAT SPECIFICATION

EXCLUDE Exclude the character from the BCC/CRC.

TRANSP Enter transparent mode.

7.4.3 Protocol Procedure Specification

Language constructs are required to specify the procedural aspects of the protocols. Apart from the constructs specifically designed for the communications environment, the functions required are similar to those of any high-level algorithmic language (assignments, conditional statements, etc). Since there are many 'flavours' of high-level languages (typified by the syntactic differences between PL/1, ALGOL 68 and Pascal), this section concentrates on the semantics of the special communications-oriented constructs with lesser emphasis on 'syntactic sugar'. These constructs are used to describe the internal structure and state transitions of the protocol processes, message queueing, and input-output operations.

7.4.3.1 Process Structure

The protocol process structure is comprised of two components, the data definitions and the procedural or algorithmic processes (Fig. 7.10).

The data definitions specify the format (memory layout) of packed data structures. Each structure is related to a particular context, which is one of:

HEADER a message header format
LINE global working variables for the protocol
STATION local working variables for each station on the line

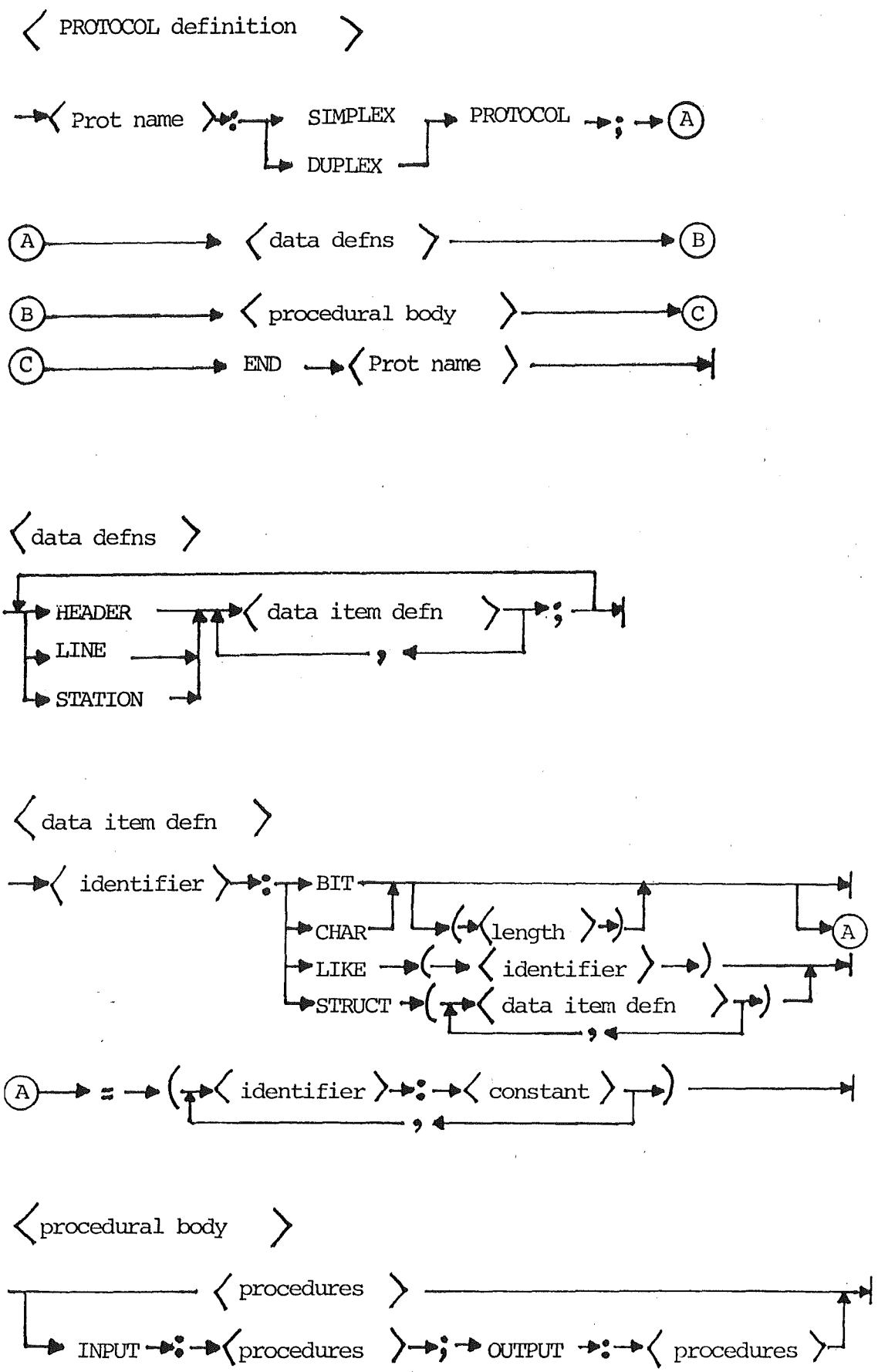


FIG. 7.10 PROTOCOL PROCESS SYNTAX

Each data item may have the following attributes:

BIT(*n*) the data item contains *n* bits
CHAR(*n*) the data item contains *n* characters (bytes)
LIKE(*id*) the data item has the same attributes as data item *id*
STRUCT(..) the data item is a structure comprised of a group of member data items

Elementary data items may have identifiers equated with particular item values.

The procedural or algorithmic specification is dependent on whether the protocol is simplex or duplex. Simplex protocols only require a single process to control the line, while duplex protocols require independent input and output processes.

7.4.3.2 Process Control

At any instant in time a protocol process may be considered to be waiting on an event, or performing an action corresponding to an event. The protocol process algorithms are expressed in terms of this state-transition model, using the constructs illustrated in Fig. 7.11. The semantics of each individual construct are discussed in following sections.

Three waiting states are defined for a process, which are terminated by the occurence of specific events.

RECEIVE Initiates a receive operation and waits until a message is received or an error condition occurs.

FETCH Fetches the next message from the station output queue.

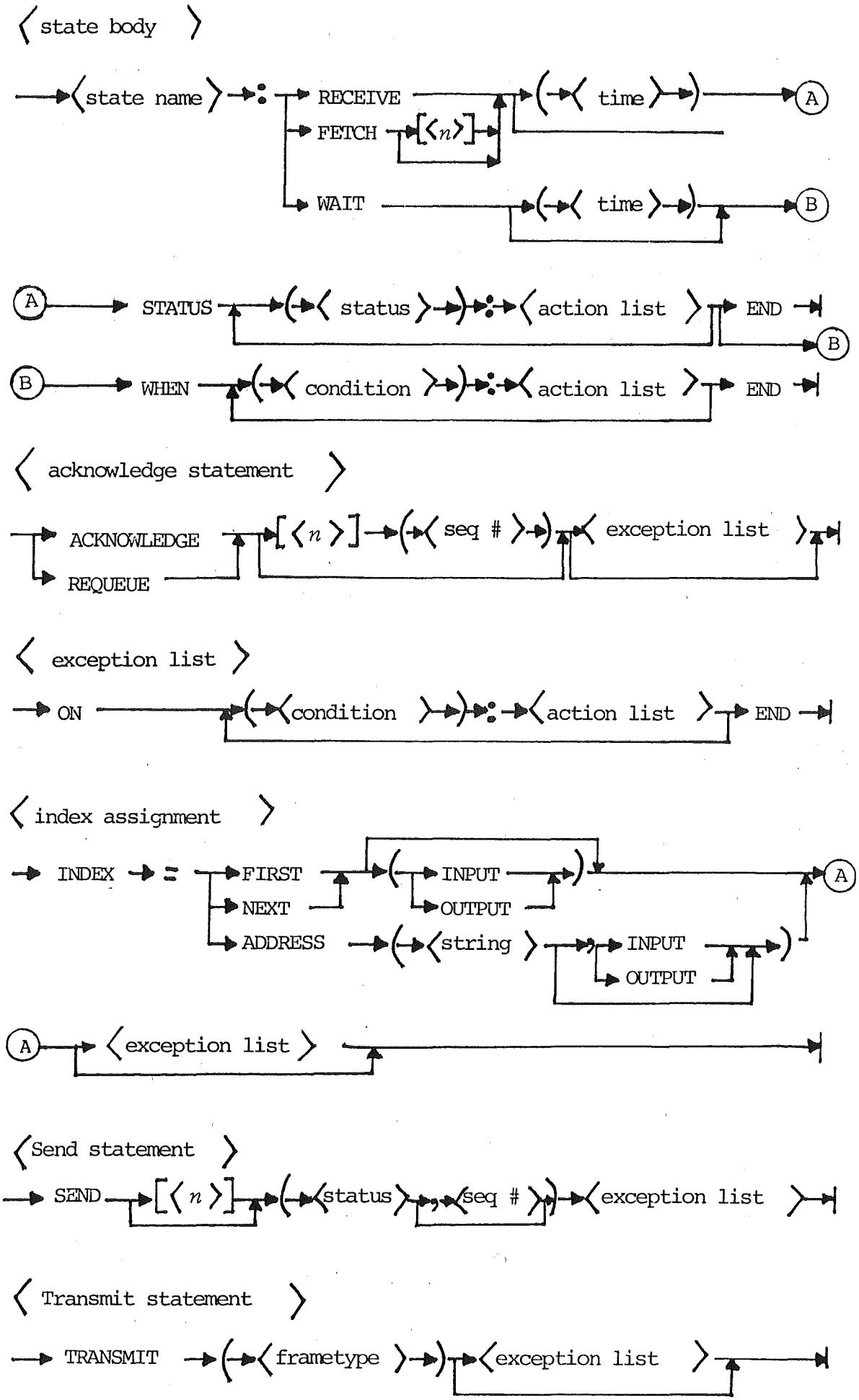


FIG. 7.11 PROTOCOL PROCESS STATEMENTS

WAIT — Delays the process for a specified period, or until a synchronising condition occurs in a duplex protocol.

When an event occurs which reactivates the process, the action corresponding to the event is selected from the set of possible actions and executed. Each action terminates by transferring to the next wait-state.

Actions are divided into two groups, depending on whether they are selected by the completion status of the RECEIVE or FETCH operation, or on the occurence of some abnormal event, such as timeout, error, or synchronising condition. These are specified by the STATUS and WHEN sets of actions respectively.

7.4.3.3 Queue Control Statements

The FETCH and SEND statements implement the interface to Session Control, performing the transfer of control status and data messages over the interface.

The types of control status are:

CONNECT — Indicates an initial connection request.

CONNACK — Acknowledges a connection request.

DISCON — Indicates a disconnection request, or the rejection of a connection request.

DISCACK — Indicates the completion of a disconnection phase.

ERROR — Reports an unrecoverable link error to Session Control.

SUSPEND — Indicates a 'Receive Not Ready' condition over the interface.

RESUME Indicates a 'Receive Ready' condition over the interface.

The types of data status are:

DATA Indicates a complete message.
BLOCK Indicates a message segment.

Conditions which may be reported are:

TIMEOUT Indicates that a message was unable to be obtained by a FETCH during the specified period.
NOTRDY Indicates a 'Not Ready' status over the interface when attempting to SEND a message to Session Control.
SEQERR Indicates a message sequence error on a SEND statement.

Protocols which use sequence numbering of messages for flow control and error detection specify the numbering modulus on the FETCH and SEND statements to determine the actual message sequence number.

The ACKNOWLEDGE statement acknowledges messages which have previously been FETCH'd from the output queue. It may be used to acknowledge all outstanding messages, or only messages up to a specified sequence number. An invalid sequence number causes a SEQERR condition.

The REQUEUE statement requeues unacknowledged messages for retransmission. It may be used to requeue all outstanding messages, or only those from a specified sequence number.

The INDEX assignment statement is used to select the current station. The station may be selected by the attributes of

whether output messages are queued, or input requests are outstanding. The order in which stations are selected is determined by:

FIRST	the highest priority station.
NEXT	the next (cyclic) station.
ADDRESS	the station corresponding to a specified address.

7.4.3.4 Input-Output Statements

The RECEIVE and TRANSMIT statements are used to initiate the input and output of messages by the Line Modules.

The TRANSMIT statement initiates transmission of the specified frame type. If an error occurs during the transmission, an exception status is reported for corrective action.

The RECEIVE statement initiates a receive operation. If a good frame is received, the corresponding STATUS action is performed. Any error returns the corresponding exception condition, which may be:

BUFFER	a buffer overflow, over-run or under-run occurred.
ECCERR	a parity, BCC or CRC error was detected.
CARRIER	a loss-of-carrier was detected.
BREAK	a break on the line was detected.
FMTERR	a character framing format error was detected.
MSGERR	a message format error was detected due to an invalid character sequence.

The INITIATE RECEIVE and INITIATE TRANSMIT statements have different semantics depending on whether the protocol is simplex or duplex. On simplex links they initiate line turn-around operations, whereas on duplex links they are used to reactivate a waiting process.

8. IMPLEMENTATION AND CONCLUSIONS

From the possible alternative techniques considered to implement the Network Interface Machine considered in Section 4.4, the use of a program (or system) generator for multi-microprocessor hardware was selected as the optimal method. This chapter describes the structure and development status of the NIM compiler, evaluates the NIM concept, and suggests areas requiring further research and development.

8.1 COMPILER STRUCTURE

The NIM compiler is separated into two distinct components, the network specification analyser and the node code generator. Utilities are also required to load the code into, and dump the code from, the processors.

8.1.1 Network Specification Analyser

The network specification analyser is based on a table-driven LR(1) parser. It analyses the network specification, producing:

- A single file containing tables describing the global network configuration.
- A file for each node containing:
  - *tables* describing the physical and logical configuration of the node,
  - *macros* corresponding to each statement of the protocol algorithms.

To simplify the implementation of the parser and reduce the syntax table size, the configuration statement attributes are specified in separate tables which are interpreted only by the lexical analyser and semantics routines.

8.1.2 Node Code Generator

The node code generator is invoked separately for each node within the network. It interprets the file produced by the network specification analyser to generate a customised operating system for each node in three phases:

- assembling the operating system modules required to support the specified hardware configuration and executive functions,
- assembling the NIM modules and tables required to implement the Session Control and Data Presentation functions,
- translating the message format specifications into decision tables, and the protocol macros into the corresponding protocol process code, to implement the Access Interfaces.

The node code generator produces a file which contains object code for each node, and loading information specifying how the code is to be loaded into the microprocessors.

8.1.3 Loader and Dump Utilities

Utility programs are required to load the object code produced by the node code generator into the communications processor, and to dump the contents of the communications processor memory and registers for problem analysis. These

utilities may operate stand-alone on the communications processor, loading the code from some physical medium such as punched tape or cassette, or may also require utilities on the host system for down-line loading and up-line recall of the object code.

8.2 IMPLEMENTATION STATUS

The portability, and to some extent, the generality, of the NIM compiler is largely determined by the implementation language. Of the high-level languages in common use today, Pascal appeared to be the most suitable since it aids the engineering of well-structured software, is portable, and available on a wide variety of computers ranging from micro-processors to mainframes. However, since the Pascal compiler for the University's Burroughs B6700 computer was still undergoing development, it was decided to write the initial implementation in Burroughs Extended Algol, the standard language of the machine.

At the time of writing, the lexical analyser and parser of the specification compiler are fully implemented and working, together with the basic code producing functions of the code generator. Although much of the NIM operating system functions have been written, these are yet to be integrated into the code-generator, together with the corresponding semantic functions of the specification compiler. The effort required to complete a project of this magnitude should not be underestimated.

8.3 CONCLUSION

This thesis has explored the concept of the Network Interface Machine, and examined and documented the requirements. From these requirements the architecture, software structure,

implementation approach and specification language have been developed. Although the design has not been validated by a completed implementation and practical experience, it is believed that the concepts and design approach are valid, and further development of the formal specification of communications protocols to complete the language design for specifying protocols is justified, together with a completed implementation.

REFERENCES

2.1 *O.S. TCAM Concepts and Facilities*
IBM Form No. GC30-2022.

2.2 *VTAM Concepts and Planning*
IBM Form No. GC27-6998.

2.3 Binder, R. et.al. 'Aloha Packet Broadcasting - A Retrospect.'
in *AFIPS Conference Proc. National Computer Conference 1975* 44:203-215.

2.4 McFadyen, J.H. 'System Network Architecture : An Overview'
IBM Systems Journal 15, No.1 (1976).

2.5 *Distributed Communications Architecture, Systems Description.*
Sperry Univac UP-8469.

3.1 *DECNET - Digital Network Architecture, Design Specification For Data Access Protocol.*
Digital Equipment Corporation (1975).

3.2 See (2.4).

3.3 *TELCON System Description.*
Sperry Univac UP-8455.

3.4 'ISO Reference Model of Open Systems Interconnection, Version 4.'
ISO/TC97/SC16 N 227 (1979).

3.5 'Draft CCITT Recommendation X.25, Interface Between Data Terminal Equipment (DTE) and Data Circuit-Terminating Equipment (DCE) For Terminals Operating in the Packet Mode on Public Data Networks.'
CCITT APVI-No. 55B, Geneva (1976).

3.6 'Revised Draft CCITT Recommendation X.25'
Computer Communications Review 10, Nos.1 & 2 (Jan/Apr 1980): 56-129.

3.7 Thurber, K.J. and Freemen, H.A. 'A Bibliography of Local Computer Network Architectures.'
Computer Communications Review 9 , No. 2 (April 1979): 1-6.

3.8 Metcalfe, R.M. and Boggs, D.R. 'Ethernet: Distributed Packet Switching for Local Computer Networks.'
CACM 19, No. 7 (July 1976): 395-404.

3.9 Thornton, J.E. et.al. 'A New Approach to Network Storage Management.'
Computer Design 14, No. 11 (Nov 1975): 81-85.

3.10 Jones, A.K. and Shwarz, P. 'Experience using Multi-processor Systems.'
Computing Surveys 12, No. 2 (June 1980): 121-165.

3.11 West, A. and Davidson, A. 'Cnet - A Cheap Network for Distributed Computing'
Computer Systems Laboratory, Queen Mary College TR120 (Mar 1978).

3.12 Davidson, A. 'Design of a Low Cost Broadcast Packet Transmission Network'
Computer Systems Laboratory, Queen Mary College TR119 (Mar 1978).

3.13 Neri, G, Morling, R.C.S. and Gau, G.D. 'A Reliable Control Protocol for High Speed Packet Transmission'
IEEE Transactions on Communications 25, No. 1 (Oct 1977): 1203-1210.

3.14 Faldella, E. Neri, G. and Salomen, T. 'Cost Effective Station Implementations in MININET.'
Euromicro Journal 5 (1979): 285-294.

3.15 *B6700 Network Definition Language Information Manual*
Burroughs Form No. 5000078.

3.16 *B7000/B6000 Series NDL Reference Manual*
Burroughs Form No. 5001522.

6.1 *ANSI Documents X3.41-1974 & X3.64-1977*
American National Standards Institute, New York.